НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 51-78

До захисту допущено
В. о. завідувача кафедри ММСА
О.Л.Тимощук
«___» _____ 2020 р.

Магістерська дисертація
на здобуття ступеня магістра за спеціальністю 122 Комп’ютерні науки
на тему: «Побудова нейронної мережі моделі класифікації для оцінки політики агента в глибокому навчанні з підкріпленням на прикладі гри Minecraft»

Виконала:
студентка ІІ курсу, групи КА-92мпв мн
Мосійчук Яна Василівна _____

Керівник: професор кафедри ММСА
д.т.н. проф. Зайченко Ю.П. _____

Рецензент:
старший науковий співробітник кафедри
програмного забезпечення комп'ютерних систем
КПІ ім. Ігоря Сікорського д.т.н., с.н.с, *Вішталь Д.М.* _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____

Київ
2020

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність — 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2020 р.

ЗАВДАННЯ
на магістерську дисертацію студентці Мосійчук Яні Василівні

1. Тема дисертації: «Побудова нейронної мережі моделі класифікації для оцінки політики агента в глибокому навчанні з підкріпленням на прикладі гри Minecraft», науковий керівник дисертації Зайченко Юрій Петрович д.т.н., затверджені наказом по університету від "21" грудня 2020 № 3414-с 011220.

2. Термін подання студентом дисертації: 13 грудня 2020 р.

3. Об'єкт дослідження: зображення, їх класифікації на фото та відео гри Minecraft

4. Предмет дослідження: методи математичного моделювання та оптимізації, які застосовуються в задачах глибинного навчання з підкріпленням

5. Перелік завдань, які потрібно розробити:

1) провести аналіз методів штучного інтелекту для вирішення прикладних задач;

2) дослідити сучасний стан та особливості застосування математичного моделювання та побудови нейронної мережі в задачах класифікації зображення;

3) побудувати нейронну мережу для задачі класифікації зображення;

4) застосувати побудовані моделі для оцінки політики агента з підкріпленням на прикладі гри Minecraft;

5) розробити стартап-проект виведення на ринок результатів дослідження;

6) розробити концептуальні висновки за результатами наукового дослідження;

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

1). Схеми побудови класичних методів класифікації (Байєса, метод опорних векторів, дерева рішень) (рис.);

2). Графічні зображення нейронних мереж (рис.);

3). Схема побудованої Q функції(рис.);

4). Приклади функціонування створеного програмного продукту (рис.);

5). Таблиці порівняльного аналізу методів класифікації

7. Дата видачі завдання: 05 вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації
1.	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів	02.09.2020 —08.09.2020
2.	Перший розділ. Чіткі моделі класифікації	09.09.2020 —15.09.2020
3.	Другий розділ. Глибинне навчання з підкріпленням	16.09.2020 —22.09.2020
4.	Третій розділ. Методика навчання згорткової нейронної мережі	23.09.2020—29.09.2020
5.	Четвертий розділ. Оцінка якості роботи класифікатора для оцінки політики агента	30.09.2020—06.10.2020
6.	П’ятий розділ. Стартап-проект	07.10.2020—13.10.2020
7.	Концептуальні висновки. Перспективи розвитку отриманих рішень	14.10.2020—20.10.2020

Студентка Я.В.Мосійчук

Науковий керівник дисертації Ю.П.Зайченко

РЕФЕРАТ

Магістерська дисертація: 84 с., 19 рисунків, 15 таблиць, 30 джерел.

В роботі розглянуті і проаналізовані одні з найбільш вживаних з тих, що існують на даний момент, сучасних методів інтелектуального аналізу даних. Проведено дослідження відомих методів класифікації, а також ефективності використання ансамблів базових класифікаторів. Окрім цього, була запропонована модель нейронної мережі для класифікації та використання її для навчання Q функціїї, доведена її ефективність на практичній задачі, а саме класифікації для гри Minecraft.

В роботі було розглянуто загальні відомості про машинне навчання, розглянуто основні складові методики Q-Learning. Було виконано аналіз сучасного використання модифікацій Q-Learning.

Об’єктом дослідження є фото та відео дані з гри Minecraft а також їх навігація та розмічені сегменти.

Предметом дослідження є математичні моделі інтелектуального аналізу даних та їх ансамблів для проведення класифікації на основі статистичних даних.

КЛАСИФІКАЦІЯ, МАШИННЕ НАВЧАННЯ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ПОПЕРЕДНЯ ОБРОБКА ДАНИХ, БЕГГІНГ, БУСТИНГ, НЕЙРОННІ МЕРЕЖІ, Q ФУНКЦІЯ

ABSTRACT

Master’s thesis: 84 pages, 19 figures, 15 tables, 30 sources.

Theme: Deep reinforcement learning for Minecraft, image classification. The known methods of classification, as well as the efficiency of use of ensembles of basic classifiers, are conducted. In addition, a neural network model was proposed to classify and use it to train Q functions, and proved its effectiveness in a practical task, namely, classification for the Minecraft game.

The paper considered general information about machine learning, the basic components of the methodology Q-Learning. The using of modern versions of Q-Learning was analyzed, such as Fuzzy Q-learning.

The subject of the study is photos and video data from the Minecraft game, as well as their navigation and marked segments.

The subject of research is mathematical models of data mining and their ensembles for classification based on statistics.

CLASSIFICATION, MACHINE LEARNING, BACKGROUND TRAINING, PRE-DATA PROCESSING, BAGGING, BUSTING, NEURAL NETWORKS, Q FUNCTION

ЗМІСТ

ВСТУП

Глибоке навчання з підкріплення (DRL) було в центрі одних з найбільших проривів штучного інтелекту (AI) за останні кілька років. Його можливості застосовуються для широкого спектра задач, від рішень щодо поліпшення сервісу для клієнтів, до передбачення захворювань COVID-19. Окремо варто згадати такий популярний напрямок, як робототехніка.

Використання алгоритмів машинного навчання для комп'ютерних ігор також набирає величезну популярність, так як будь-яка відеогра є відмінним тестовим полігоном для алгоритмів, що використовуються при створенні моделей штучного інтелекту. Багато відомих корпорації такі як Google і Microsoft тестують і покращують свої моделі ШІ на комп'ютерних іграх.

Не маючи жодного уявлення про гру, чи може штучний інтелект навчиться ефективно грати в неї? У даній роботі розглядається проблема створення штучного інтелекту для динамічного середовища про яку агент нічого не знає. В ролі динамічного середовища, виступає комп'ютерна гра Minecraft.

Для створення моделі використовується глибоке навчання з підкріпленням. Ідея такого підходу бере свій початок в психології і вивчає те, як деякий "агент" повинен діяти в рамках певного середовища, щоб максимізувати "виграш". Такі алгоритми намагаються знайти стратегію дій агента, визначаючи пару стан оточення - дійства, які необхідно вжити для отримання вигоди.

Minecraft - це 3D-гра, що відкриває світ від першої особи, орієнтована на збирання ресурсів та створення структур та предметів. Ці структури та предмети мають необхідні інструменти та матеріали для їх створення. Процедурно генерований світ складається з дискретних блоків, які дозволяються модифікувати. У процесі гри, гравці змінюють своє оточення, збираючи ресурси та будуючи структури. Мета агента - отримати алмаз

(винагороду). Агент починає свій рух у випадковому вихідному місці без будь-яких предметів і отримує винагороду за отримання предметів.

Метою цієї роботи є дослідження ефективності згорткової нейронної мережі в задачі класифікації зображень в порівнянні з методами штучного інтелекту, в навчанні з підкріпленням у середовищі гри Minecraft.

Об’єктом дослідження магістерської дисертації є інтелектуальна відеогра MineCraft.

Предметом дослідження є методи інтелектуального аналізу та класифікації даних, що використовуються для пошуку оптимальної поведінки агента в грі MineCraft.

В ході роботи було поставлено та вирішено наступні задачі:

1. Дослідження існуючих підходів для вирішення задач класифікації зображень.
2. Побудова моделей класифікації на основі методів опорних векторів, наївного байєсівського класифікатора та дерев рішень.
3. Проектування та розробка архітектури згорткової нейронної мережі для класифікації зображень.
4. Аналіз результатів класифікації зображень для оцінки політики агента в глибокому навчанні з підкріпленням на прикладі гри Minecraft.

РОЗДІЛ 1. ВВЕДЕННЯ У ПРЕДМЕТНУ ОБЛАСТЬ

Постановка задачі:

1. Провести дослідження існуючих підходів класифікації зображення.
2. Дослідити та вивчити основні принципи роботи методики машинного навчання Q-Learning.
3. Підібрати навчальну та тестову вибірку на основі знімків з гри Minecraft.
4. Спроектувати та побудувати програмний комплекс, що реалізував автоматичну класифікацію предметів.
5. Розробити тестове середовище для оцінки політики агента на основі вихідних даних після класифікації зображення.

1.1 Актуальність теми

Не так давно світ спостерігав, як чемпіон світу з шахів Гаррі Каспаров програв вирішальний матч проти суперкомп'ютера. Deep Blue від IBM втілив сучасний стан в кінці 90-х, коли машина, яка перемагала світового (людського) чемпіона в такій складній грі, як шахи, ще була нечувана. Швидко перемотуючись на сьогодні, не тільки суперкомп’ютери значно перевершили гру у шахи, їм вдалося досягти надлюдських результатів у низці інших ігор, починаючи від Go to Dota і закінчуючи класичними назвами Atari. Багато з цих ігор були освоєні лише за останні п'ять років, вказуючи на темпи інновацій набагато швидше, ніж за два десятиліття до цього. Нещодавно Google випустив роботу над Agent57, яка вперше продемонструвала чудову продуктивність порівняно з існуючими тестами у всіх 57 іграх Atari 2600.

Клас алгоритмів ШІ, що лежать в основі цих подвигів – глибинне навчання з підкріпленням (Deep Reinforcement Learning, DRL) -

продемонстрував здатність вчитися на дуже високих рівнях у обмежених доменах, таких як ті, що пропонуються іграми. Подвиги в іграх дали цінну інформацію (для дослідницької спільноти) про те, що глибоке навчання з підкріпленням може, а що не може робити. Запуск цих алгоритмів вимагав гігантської обчислювальної потужності, а також тонкої настройки нейронних мереж. Дослідники застосовують нові підходи, такі як мульти-середовище навчання та використання мовного моделювання, щоб допомогти у навчанні в різних сферах.

Глибинне навчання з підкріпленням вже показало велику ефективність у обмежених середовищах, що стимулювало його використання в таких сферах, як робототехніка та охорона здоров’я. У робототехніці DRL показав багатообіцяючі результати у використанні середовищ моделювання для підготовки роботів до реального світу. Інші сфери, в яких можливе ефективне використання DRL, включають обробку природних мов, комп'ютерний зір, алгоритмічну оптимізацію та фінанси.

1.2 Вхідні дані

Minecraft — одна з найуспішніших комп’ютерних ігор в історії. Станом на 2020 рік її сукупні продажі по всьому світу перевищили 200 млн копій, а кількість унікальних гравців — 85 млн. Компанія Microsoft створила освітню версію гри якою користуються в школах по всьому світу. Гра була створена в 2011 році, гравець подорожує тривимірним світом, що складається з кубічних блоків. Він може вільно перебудовувати його, створюючи з блоків складні споруди. Це робить Minecraft схожою на конструктор LEGO. Тобто Minecraft це тривимірний процедурно генерований світ (Рисунок 1.1).

Рисунок 1.1 – Картинка з гри MineCraft

У Minecraft гравець має втілення у вигляді персонажа. Гра закінчується коли гравець (агент) знаходить скарб (diamond).

Набір даних це зображення з гри (назва датасету - MineRL-v0) був опублікований компанією Microsoft разом з Google у 2019 році для вирішення двох задач:

1. Розпізнавання та класифікація об’єктів на картинці.
2. Побудова моделі з цілю навчити агента знаходити скарб серед усіх об’єктів гри.

Набір даних складає майже 60 мільйонів картинок з реальних ігор і задача агента навчити розпізнавати об’єкти аби знайти скарб (diamond, один з класів). Для дослідження обрано десять класів (building, structures, road, cars, dog, cat, bucket, block, Minecraft kelp та diamond як скарб) (Рисунок 1.2);

Bad

Diamond

Dog

Рисунок 1.2 - Приклади зображень з мітками класів

1.3 Висновки до першого розділу

У першому розділі розглянуто актуальність теми, останні тенденції в глибинному навчанні з підкріпленням. Описані вхідні дані для експерименту на прикладі гри Minecraft, наведені приклади зображень з мітками а також класи для котрі були обрані для симуляції гри.

РОЗДІЛ 2. ЧІТКІ МЕТОДИ КЛАСИФІКАЦІЇ

Машинним навчанням називається галузь комп'ютерних наук, яка вивчає методи навчання комп'ютеризованих систем на підставі даних без програмування їх поведінки [1]. Методи машинного навчання (machine-learning methods) відіграють важливу роль у багатьох аспектах сучасного суспільства: від веб-пошуку до фільтрації контенту в соціальних мережах. Системи на базі методів машинного навчання використовуються в системах машинного зору, для ідентифікації об'єктів на зображеннях, аналізу людської мови і текстів тощо [2, 3]. Традиційно розпізнавання образів (або їх класифікація) здійснювалось на основі інформаційних ознак. Отже, побудова систем розпізнавання образів (pattern-recognition) або систем, в основу яких покладено методи машинного навчання, потребувала експертних знань для розроблення методів та правил виокремлення ознак (feature extraction). Виокремлення ознак — це перетворення початкових «сирих» даних (таких як значення піксела на зображенні) у придатне подання (вектор ознак), з якого система навчання (класифікатор) може виявити і класифікувати образи, що подаються на вхід. Такі методи машинного навчання обмежені в можливостях обробляти природні дані в початковому вигляді [4].

Важливою властивістю класифікаторів є можливість не лише належності вхідних даних до певного класу (виходу класифікатора), а і визначення ймовірності належності до кожного з класів, на основі якої легко обрати найбільш достовірний клас [5]. Таку особливість має, наприклад, логістична регресія (logistic regression). Для вирішення задач класифікації найчастіше використовують: штучні нейронні мережі (artificial neural network) [1], логістичну регресію [1], метод опорних векторів (Support Vector Machine (SVM)) [1], дерево рішень (random forest) [8] та Наївний байєсівський класифікатор.

2.1 Метод опорних векторів (SVM)

У цьому алгоритмі ми побудуємо кожен елемент даних як точку в n-мірному просторі (де n - кількість ознак), причому значення кожної ознаки є значенням певної координати. Потім ми проводимо класифікацію, знаходячи гіперплощину, яка дуже добре розмежовує два класи Рисунок 2.1.

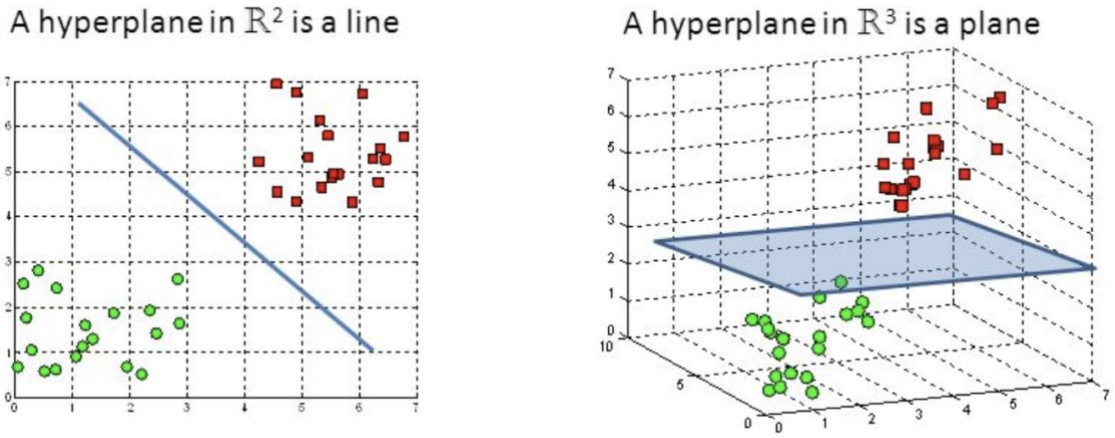

Рисунок 2.1 - Метод опорних векторів

Щоб розділити два класи точок даних, можна вибрати багато можливих гіперплощин, мета - знайти площину, яка має максимальний запас, тобто максимальну відстань між точками даних обох класів. Максимізація відстані поля забезпечує деяке підкріплення.

Гіперплощини - це межі рішення, які допомагають класифікувати точки даних. Точки даних, що падають з будь-якої сторони гіперплощина, можна віднести до різних класів. Також розмірність гіперплощини залежить від кількості ознак. Якщо кількість вхідних функцій дорівнює 2, то гіперплощина - це лише лінія. Якщо кількість вхідних ознак дорівнює 3, то гіперплощина стає двовимірною площиною (Рисунок 2.2).

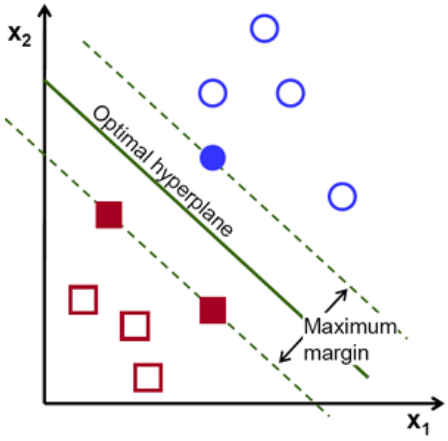

Рисунок 2.2 - Вибір гіперплощини

В алгоритмі SVM задача домогтись максимальної межі між точками даних та гіперплощиною. Функція втрат, яка допомагає максимізувати маржу (Рисунок 2.3).

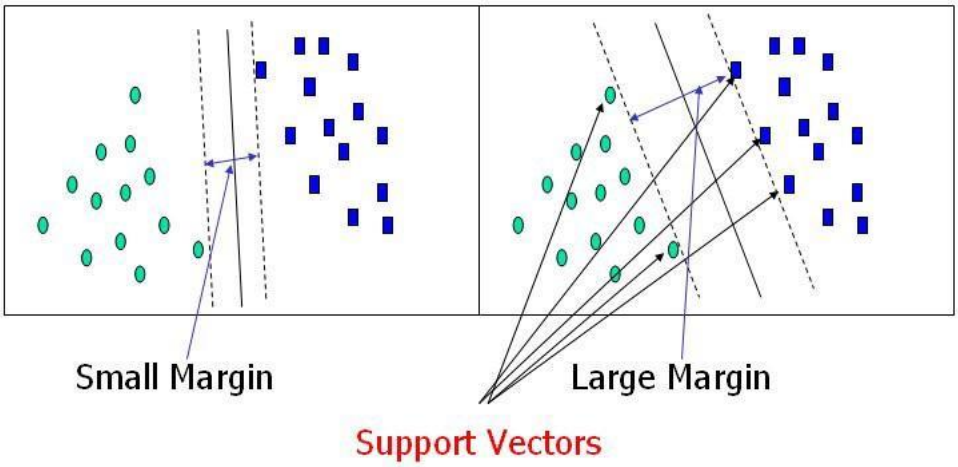

Рисунок 2.3 - Маржа в методі SVM

Мета SVM - навчити модель, яка призначає нові невидимі об’єкти до певної категорії, створюючи лінійний розділ простору зображень на дві категорії. Виходячи з особливостей у нових невидимих об'єктах (наприклад, документи / електронні листи), він розміщує об’єкт "вище" або "нижче"

площини розділення, що призводить до категоризації (наприклад, спаму чи неспаму). Це робить його прикладом неімовірнісного лінійного класифікатора.

Алгоритм SVM

Вхідні дані: набір (вхідних, вихідних) зразків навчальної пари; вхідний зразок має x1, x2… xn, а вихідний результат y.

Вихідні дані: набір вагових коефіцієнтів w (або wi), по одному для кожної функції, лінійна комбінація якої передбачає значення y.

Визначимо гіперплощини H таким чином, щоб:

$w * xi + b \geq +1$, коли $yi = + 1$

$w * xi + b \leq -1$, коли $yi = –1$

d + найкоротша відстань до найближчої позитивної точки

d- найкоротша відстань до найближчої негативної точки

Маржа розділюючої гіперплощини дорівнює d+ + d–. Алгоритм оптимізації для формування ваг здійснюється таким чином, що лише опорні вектори визначають ваги і, отже, межі. Рівняння, що визначає розділення гіперплощини:

$wTx + b = 0$

- w - вектор ваги
- x - вхідний вектор
- b - шум

$wTx + b \geq 0$ для $di = +1$

$wTx + b < 0$ для $di = –1$

Перевага SVM полягає в тому, що вони не обмежуються лінійними класифікаторами. Використовуючи техніку, відому як трюк ядра, вони можуть стати набагато гнучкішими, вводячи різні типи нелінійних меж рішення.

Формально в математичній мові SVM будують лінійні розділюючі гіперплани у великомірних векторних просторах. Точки даних розглядаються як ($\vec{x}$, y) кортежі, $\vec{x} = (x1,…, xp)$, де xj - значення ознак, а y - класифікація

(зазвичай дається як +1 або −1). Оптимальна класифікація відбувається тоді, коли такі гіперплани забезпечують максимальну відстань до найближчих точок навчальних даних.

Однак якщо в просторі функцій деякі множини не є лінійно відокремленими (тобто вони перетинаються), то необхідно виконати зіставлення вихідного простору властивостей у просторі більш високого розміру, в якому поділ між групами є чіткий, або принаймні зрозуміліший. Однак це призводить до того, що робить межу поділу у вихідному просторі потенційно нелінійною [2].

Щоб знайти гіперплан максимальної границі потрібно: по-перше, обчислити перпендикулярну відстань від кожного навчального спостереження $\vec{x}$ і для заданої роздільної гіперплани. Найменша перпендикулярна відстань до тренувального спостереження від гіперплану відома як межа.

Метод опорних векторів SVM (Support Vector Machine) можна розглянути як нелінійне створення лінійного класифікатора, на основі розширення розмірів вихідного простору за допомогою спеціальних ядерних функцій. Це дозволяє будувати моделі з використанням розділяючих площин самої різної форми.

Власне ядром є будь-яка симетрична, позитивно напів-визначена матриця K, яка складена з скалярних добутків пар векторів $\boldsymbol{x}_i$ та $\boldsymbol{x}_j$:$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_j) \rangle$ характеризують міру їх близькості. Тут φ - довільна функція, що формує ядро. В якості таких функцій найчастіше використовують такі:

- Лінійне ядро: $K(\boldsymbol{x}_i,\boldsymbol{x}_j) = \boldsymbol{x}_i,\boldsymbol{x}_j$, що відповідає класифікатору на опорних векторах в вихідному просторі.
- Поліноміальне ядро зі степеню p: $K(\boldsymbol{x}_i,\boldsymbol{x}_j)=(1+\boldsymbol{x}_{Ti}\boldsymbol{x}_j)^{\wedge}p$.
- Гауссове ядро (RBF): $K(\boldsymbol{x}_i,\boldsymbol{x}_j)=\exp(\gamma||\boldsymbol{x}_i-\boldsymbol{x}_j||^{\wedge}2)$.
- Сигмоїдне ядро : $K(\boldsymbol{x}_i,\boldsymbol{x}_j)=\tanh(\gamma\boldsymbol{x}_{Ti}\boldsymbol{x}_j+\beta_0)$.

Кожне ядро характеризується параметрами (p, γ, β0 і т.д.), які підлягають оптимізації. Основна ідея використання ядер полягає в тому, що при відображенні даних у простір більш високої розмірності вихідна кількість точок може стати лінійно нероздільні (Cristianini, Shawe-Taylor, 2000). Тоді природно припустити, що оптимальна розділяюча гіперплоскость методу опорних векторів може бути знайдена шляхом підбору коефіцієнтів. Таким чином, як і в лінійному випадку, вирішується математично зручне завдання квадратичної оптимізації з використанням множників Лагранжа. Обчислення екстремуму в розширених просторах настільки величезних розмірностей виявилося також можливим тому, що ядро формується тільки для обмеженого набору опорних векторів.

Перевагами методу опорних векторів:

- збіжність до глобального мінімум;
- ефективність за обмеженої кількості навчальних даних [5].

Недоліки методу:

- модель SVM відносить новий приклад до тієї чи іншої категорії, що робить її не ймовірнісною, а лише бінарним класифікатором [4];
- використання ядер забезпечує роботу з лінійно неподільними множинами даних, але не дозволяє отримувати достовірні узагальнення для векторів, що віддалені від навчальних даних;
- навчання SVM є надто повільним і не може використовуватися для аналізу великих обсягів даних [5].

2.2 Дерева рішень (Decision trees)

Дерево рішень - це деревоподібна структура, схожа на блок-схему, де внутрішній вузол являє знаки (або атрибут), гілка представляє правило прийняття рішення, а кожен кінцевий вузол являє результат. Самий верхній

вузол в дереві рішень називається кореневим вузлом. Він вчиться відділятися на основі значення атрибута. Він рекурсивно розбиває дерево, називаючи його рекурсивним розподілом (Рисунок 2.4) [6].

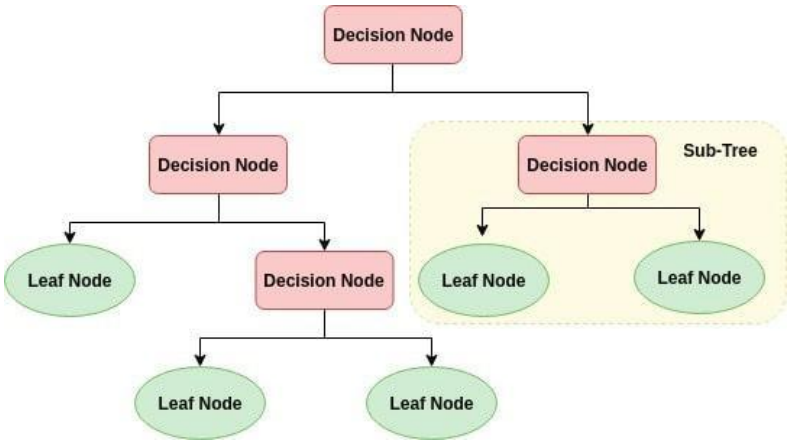

Рисунок 2.4 - Принцип побудови дерева рішень

Дерево рішень - це white box тип алгоритму ML. Він поділяє внутрішню логіку прийняття рішень, яка недоступна в black box таких алгоритмів, як нейронні мережі. Складність часу дерев рішень - це залежність від кількості записів і кількості атрибутів у визначених даних. Дерево рішень - це метод, не розподілений чи непараметричний, який не залежить від припущень щодо розподілу ймовірностей.

Алгоритм побудови дерева рішень:

1. Обирається найкращий атрибут за допомогою заходів вибору атрибутів (ASM) для розділення записів.

2. Атрибут стає вузлом рішення і розбиває набір даних на менші підмножини.

3. Побудова дерева: повторювати кроки вище рекурсивно для кожної ознаки, поки одна з умов не відповідатиме:

- усі кортежі належать до одного і того ж значення атрибута;
- більше немає атрибутів;
- більше випадків немає.

Міра вибору атрибутів є евристичною для вибору критерію розщеплення, який найкращим чином підбирає дані розділів. Він також відомий як правила розщеплення, оскільки він допомагає нам визначити точки прориву кортежів на заданому вузлі. ASM забезпечує рейтинг кожної функції (або атрибуту), пояснюючи даний набір даних. Для вибору атрибуту потрібно ґрунтовні математичні підходи. Атрибут найкращої оцінки буде обраний як атрибут поділу. У випадку з атрибутом безперервного значення також потрібно визначити точки розділення для гілок. Найпопулярніші заходи відбору - інформаційний приріст, коефіцієнт посилення та індекс Джині.

Шеннон винайшов концепцію ентропії, яка вимірює домішки вхідного набору. У фізиці та математиці ентропію називають випадковістю чи домішкою в системі. В теорії інформації йдеться про домішки в групі прикладів. Приріст інформації - це зменшення ентропії. Приріст інформації обчислює різницю між ентропією перед розділенням і середньою ентропією після розбиття набору даних на основі заданих значень атрибутів. Алгоритм дерева рішень ID3 (Iterative Dichotomiser) використовує посилення інформації[7].

$$Info(D) = -\sum_{i=1}^{m} \quad pipi \ ,$$

Pi - ймовірність того, що довільний кортеж у D належить до класу Ci.

$$Info(D) = \sum_{j=1}^{v} \quad \frac{|D_j|}{|D|}$$

$$Gain(A) = Info(D) - \ Info_A(D) \ ,$$

- Info(D) - це середній обсяг інформації, необхідний для ідентифікації класової мітки кортежу в D;
- |Dj|/|D| виступає вагою j-ої перегородки;

- InfoA(D) - очікувана інформація, необхідна для класифікації кортежу з D на основі розбиття за A.

Атрибут A з найбільшим посиленням інформації, посиленням (A), вибирається як атрибут розбиття у вузлі N().

Приріст інформації упереджений для атрибута з багатьма результатами. Це означає, що він віддає перевагу атрибуту з великою кількістю різних значень. Наприклад, атрибут з унікальним ідентифікатором, таким як customer_ID, має нульову інформацію (D) через чистий розділ. Це максимально збільшує приріст інформації та створює марний розподіл.

Алгоритм C4.5, вдосконалення ID3, використовує розширення до інформаційного посилення, відоме як коефіцієнт посилення. Коефіцієнт посилення вирішує проблему зміщення шляхом нормалізації отримання інформації за допомогою розділеної інформації[8].

$$SplitInfo_A(D) = \sum_{j=1}^{v} \quad \frac{|D_j|}{|D|} \times \frac{|D_j|}{|D|} \ ,$$

- |Dj|/|D| виступає вагою j-ої перегородки;
- v - кількість дискретних значень в атрибуті A;

Коефіцієнт посилення можна визначити як

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

Атрибут з найбільшим коефіцієнтом посилення вибирається як атрибут розбиття.

Інший алгоритм дерева рішень CART (Дерево класифікації та регресії) використовує метод Джині для створення розділених точок[9].

$$Gini(D) = 1 - \sum_{i=0}^{m} \quad Pi^2 \ ,$$

де pi - ймовірність того, що кортеж у D належить до класу Ci.

Індекс Джині вважає двійковий розподіл для кожного атрибута. Ви можете обчислити зважену суму домішок кожного розділу. Якщо двійковий розподіл на атрибут A розділів даних D на D1 і D2, індекс Джині D дорівнює:

$$Gini(D) = \frac{|D1|}{|D|} Gini(D) + \frac{D_2}{D} Gini(D_2)$$

У випадку атрибута з дискретним значенням підмножина, яка дає мінімальний індекс Джині, вибирається як атрибут розділення. Що стосується атрибутів безперервного значення, стратегія полягає у виборі кожної пари сусідніх значень як можливої точки розділення та точки з меншим індексом джині, обраним як точку розподілення[9].

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

Атрибут з мінімальним індексом Джині вибирається як атрибут розділення.

Переваги

1. Дерева рішень легко інтерпретувати та візуалізувати.
2. Алгоритм може легко знаходити нелінійні зв’язки.
3. Вимагає меншої попередньої обробки даних від користувача, наприклад, немає необхідності нормалізувати стовпці.
4. Може бути використаний для інженерних функцій, таких як прогнозування відсутніх значень, придатних для вибору змінної.
5. Дерево рішень не має припущень щодо розподілу через непараметричний характер алгоритму[10].

Недоліки

1. Чутлива до зашумлених даних. Легко перенавчається через шум в даних.
2. Невелика дисперсія даних може призвести до різного виду дерева рішень. Це може бути зменшено за допомогою алгоритмів беггінга та бустинга[10].
3. Дерева рішень чутливі до незбалансованих наборів даних[11].

2.3 Наївний байєсівський класифікатор (Naive Bayes)

Наївний байєсівський класифікатор - це методика класифікації, заснована на теоремі Байеса з припущенням незалежності серед прогнозів. Простіше кажучи, класифікатор Naive Bayes припускає, що наявність певної особливості в класі не пов'язана з наявністю будь-якої іншої функції[12].

Наприклад, яблуко можна вважати фруктом, якщо воно червоне, кругле і має діаметр близько 3 дюймів. Навіть якщо ці особливості залежать один від одного або від наявності другої функції, всі ці властивості незалежності створюють ймовірність того, що цей фрукт - яблуко, і він відомий як Наївний.

Модель Naive Bayes проста у створенні та особливо корисна для дуже великих наборів даних. Поряд із простотою, Naive Bayes, як відомо, перевершує навіть дуже складні методи класифікації.

Теорема Байєса забезпечує спосіб обчислення задньої ймовірності P (c | x) з P (c), P (x) і P (x | c), рівняння нижче[12]:

$$P(c|x) = \frac{P(c)P(c)}{P(x)} \ ,$$

- P (c | x) - ймовірність класу (c, цілі), заданого предиктором (x, атрибутами);

- P (c) - це попередня ймовірність класу;
- P (x | c) - ймовірність, яка є ймовірністю прогнозованого даного класу;
- P (x) - це попередня ймовірність предиката.

Різні наївні класифікатори Байєса відрізняються, головним чином, від припущень, які вони роблять щодо розповсюдження. Незважаючи на їх, очевидно, надто спрощені припущення, наївні класифікатори (котрі часто використовуються у класифікації) Байєса досить добре попрацювали в багатьох реальних ситуаціях, знаменито класифікуючи документи та фільтруючи спам. Вони потребують невеликої кількості даних про навчання для оцінки необхідних параметрів.

Наївні байесські учні та класифікатори можуть бути надзвичайно швидкими порівняно з більш досконалими методами. Розв'язка умовного розподілу класових умовних ознак означає, що кожен розподіл може бути незалежно оцінений як одновимірний розподіл. Це, в свою чергу, допомагає полегшити проблеми, пов'язані з прокляттям розмірності[13].

Переваги:

- Легко і швидко передбачити клас набору тестових даних. Також працює для багатокласової класифікації;
- Коли припускається незалежність, класифікатор Naive Bayes працює краще порівняно з іншими моделями, такими як логістична регресія, потрібно менше даних про навчання;
- Наївний байєсівський класифікатор добре працює у випадку категоріальних вхідних змінних порівняно з числовими змінними. Для числової змінної передбачається нормальний розподіл.

Недоліки:

- Якщо категоріальна змінна має категорію (у тестовому наборі даних), якої не спостерігали у наборі даних тренувань, то модель призначить імовірність 0 (нуль) і не зможе зробити прогнозування.

Це часто називають "нульовою частотою". Для вирішення цього питання ми можемо використовувати техніку розгладження. Один з найпростіших методів розгладження називається оцінкою Лапласа;

- З іншого боку, наївний Байєс також відомий як поганий оцінювач, тому ймовірні результати з прогнозування проби не слід сприймати занадто серйозно;
- Ще одне обмеження Naive Bayes - припущення незалежних прогнозів. У реальному житті майже неможливо, що ми отримаємо набір прогнозів, які є абсолютно незалежними[14].

Варіанти підвищення потужності моделі Naive Bayes:

- якщо є нормальний процес розподілу, безперервні функції, ви повинні використовувати багато способів, щоб співати або зміни в зв'язку зі зміною нормального розподілу;
- видаліть введені елементи належать до голосування, так як важко моделювати інші причини можуть бути дуже великими.Класифікатори Naive Bayes мають обмежені можливості налаштування параметрів, наприклад, альфа = 1 для згладжування, fit_prior = [Правда | Неправильно], щоб дізнатися попередні ймовірності класу чи ні та деякі інші параметри [15].

2.4 Використання ансамблів для задач класифікації

Одним із способів підвищення точності моделей є створення та тренування ансамблів моделей - тобто набори моделей, що використовуються для вирішення однієї і тієї ж проблеми. Під ансамблевою підготовкою розуміється підготовка скінченного набору основних класифікаторів з подальшим поєднанням результатів їх прогнозування в єдиний прогноз

зведеного класифікатора. Зрозуміло, що комбінований (агрегований) класифікатор дасть більш точний результат, особливо якщо:

- кожен з базових класифікаторів сам по собі має непогану точністю;
- вони призводять до різних результатів (помиляються на різних множинах).

Наведемо чотири причини, які виділяють експерти:

2. Статистична. Як вже було сказано, агрегований класифікатор «усереднює» помилку кожного з базових класифікаторів - відповідно, вплив випадковостей на усереднену гіпотезу істотно зменшується.

3. Обчислювальна. Наведемо приклад з реального життя: припустимо, що на якійсь обмеженій території заритий скарб. При цьому рельєф і, наприклад, рослинність цієї місцевості неоднорідні - а ще доводиться враховувати такі фактори, як обмеженість в часі і погодні умови. У нашому випадку скарб - це глобальний оптимум, і ансамбль моделей (команда «Індіан Джонсів») має більший шанс знайти його, оскільки буде шукати його з різних точок вихідного безлічі гіпотез (з різних точок нашої з вами території) [16].

4. Репрезентативна. Може трапитися і так, що агрегована гіпотеза буде знаходиться за межами безлічі гіпотез базових - в такому випадку при побудові комбінованої гіпотези ми розширимо безліч можливих гіпотез.

5. Побудувати ансамбль моделей можна з використанням декількох методів, але найпоширеніші з них наступні: бустинг та беггінг[17].

2.4.1 Бустінг

Бустінг - це метод, який полягає в послідовній адаптації декількох слабких учнів: кожна модель порівнюється послідовно, що надає велику вагу об'єктів в наборах даних, які погано оброблялися попередніми моделями в

послідовності. Інтуїтивно, кожна нова модель зосереджує свої зусилля на найбільш складних об'єктах вибірки при навчанні попереднім моделям, так що в кінці процесу ми отримуємо сильного учня з низькими змінами (навіть якщо вийде так, що бустінг буде при цьому зменшувати розкид) (Рисунок 2.5) [18].

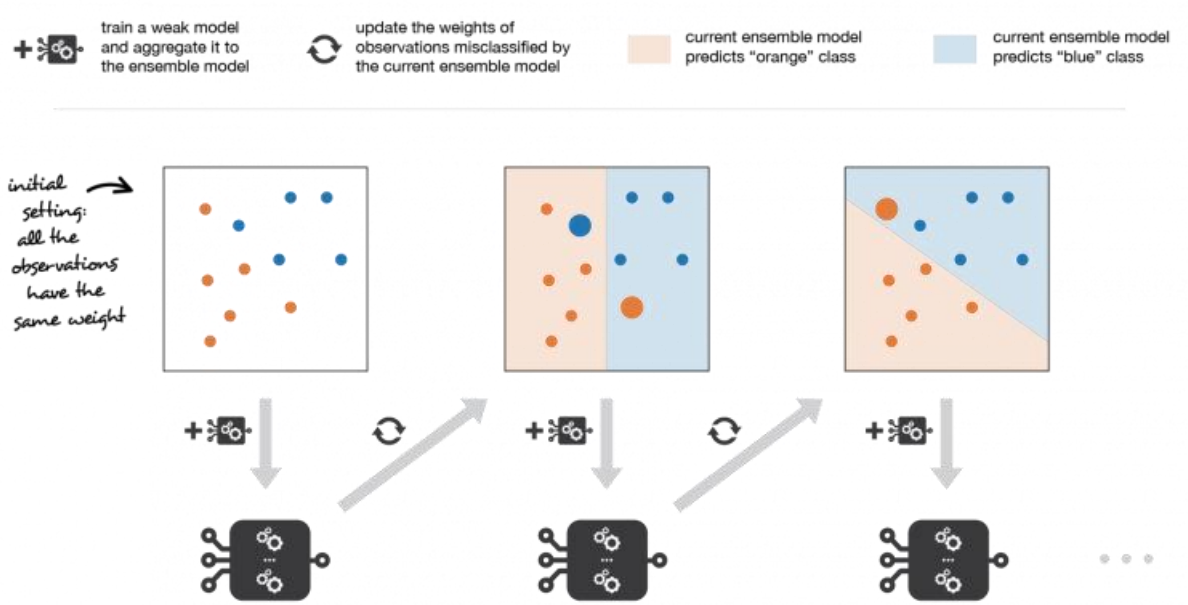

Рисунок 2.5 - Алгоритм побудови бустінга

Базові моделі, які часто розглядаються для бустінга - це моделі з низьким розкидом, але з високим зміщенням. Наприклад, якщо ми хочемо використовувати дерева рішень в якості наших базових моделей, в основному ми будемо вибирати неглибокі дерева рішень з глибиною в кілька вузлів. Інша важлива причина, яка мотивує використовувати моделі з низьким розкидом, але з високим зміщенням в якості слабких учнів для бустінга, полягає в тому, що ці моделі, як правило, вимагають менших обчислювальних витрат (кілька ступенів свободи при підборі гіперпараметров). Вважається що бустінг це одна з найдієвіших методів підвищення класифікації моделей. Дійсно, оскільки обчислення для підгонки до різних моделей не можуть виконуватися паралельно (на відміну від беггінга), це може стати занадто дорогим для послідовного підбору декількох складних моделей.

Після того, як слабша половина учнів обрані, нам все ще потрібно визначити, як вони будуть послідовно підганяти (яку інформацію з попередніх моделей ми враховуємо при підборі поточної моделі?) І як вони будуть агрегуватися. Ми обговоримо ці питання в двох наступних підрозділах, більш докладно описують два важливих алгоритму бустінга: adaboost (адаптивний бустінг) і градієнтний бустінг.

Два мета-алгоритму відрізняються тим, як вони створюють і об'єднують слабких учнів в ході послідовного процесу. Адаптивний бустінг оновлює ваги, прикріплені до кожного з об'єктів навчального датасета, тоді як градієнтний бустінг оновлює значення цих об'єктів. Ця різниця виходить з того, що обидва методи намагаються вирішити задачу оптимізації, яка полягає в пошуку найкращої моделі, яка може бути записана у вигляді зваженої суми слабких учнів[19].

2.4.2 Беггінг

На відміну від попереднього методу, беггінг (bootstrap aggregating) використовує паралельне навчання базових класифікаторів (кажучи мовою математичної логіки, беггінг - поліпшує об'єднання, а бустінг - поліпшує перетин) (Рисунок 2.6).

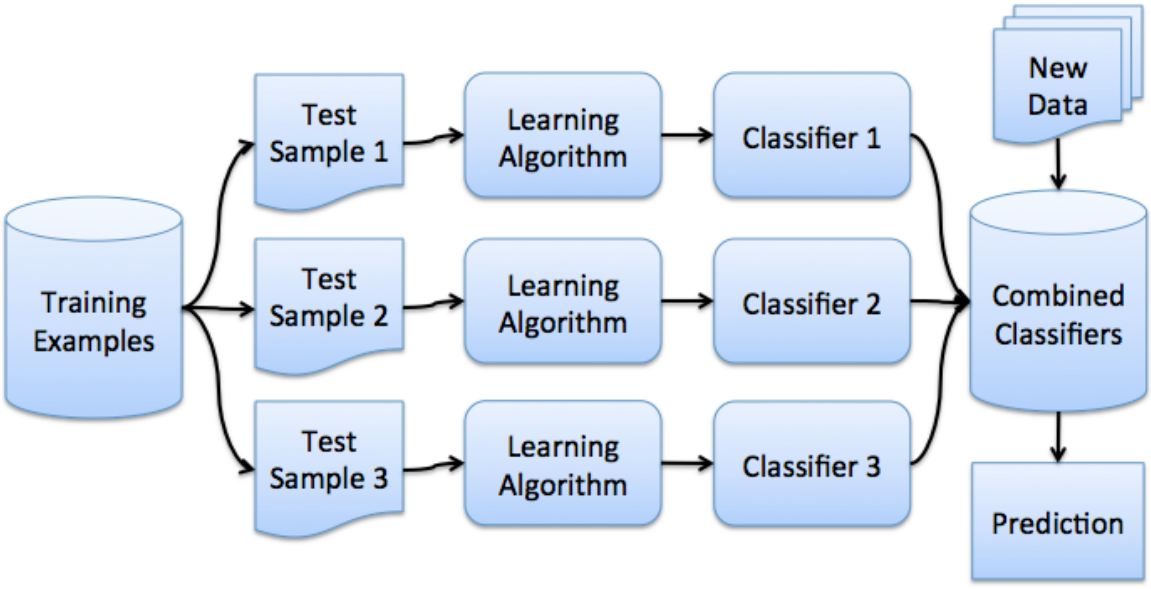

Рисунок 2.6 - Алгоритм побудови беггінга

В ході беггінга відбувається наступне:

1. З безлічі вихідних даних випадковим чином відбирається кілька підмножин, що містять кількість прикладів, що відповідає кількості прикладів вихідного безлічі.

2. Оскільки відбір здійснюється випадковим чином, то набір прикладів завжди буде різним: деякі приклади потраплять в кілька підмножин, а деякі не потраплять ні в одне.

3. На основі кожної вибірки будується класифікатор.

4. Висновки класифікаторів агрегуються (шляхом голосування або усереднення) [19].

Як і у випадку бустінгу, очікується, що результат прогнозування зведених класифікаторів буде набагато точнішим, ніж результат прогнозування окремих моделей на одному наборі даних.

2.5 Висновки до другого розділу

У другому розділі розглянута загальна задача та опис машинного навчання, в особливості задач класифікації. Були розглянуті та вивченні наступні методи класифікації: байєсівський класифікатор, метод опорних векторів, дерева рішень, наведений алгоритм побудови кожного з них а також аналіз переваг та недоліків. Наведений опис алгоритмів та методів, описана математична база кожного методу та проілюстрований принцип побудови.

Розглянуті методи покращення ефективності класифікаторів: методи створення ансамблів класифікаторів, алгоритми беггінга, бустінга, а також наведені приклади коли ці підходи спрацюють найкраще.

РОЗДІЛ 3. ГЛИБИННЕ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Глибинне навчання з підкріпленням - це категорія машинного навчання, де інтелектуальні машини можуть вчитися на своїх діях, подібних до того, як люди навчаються на досвіді. Для цього типу машинного навчання притаманно, що агент винагороджується або карається на основі його дій. Дії, що призводять до цільового результату, винагороджуються (посилюються).

Завдяки серії спроб і помилок машина продовжує навчатися, що робить цю технологію ідеальною для динамічного середовища, яке постійно змінюється. Незважаючи на те, що навчання з підкріплення існує вже десятки років, недавніше поєднувалось із глибинним навчанням, дало феноменальні результати.

Можливості глибинного навчання з підкріпленням (DRL) привернули увагу багатьох під час широко розрекламованої поразки гросмейстера Go від AlphaGo DeepMind. На додаток до гри в Go, модель досягло рівня людського рівня в інших іграх, таких як шахи, покер, ігри Atari та кілька інших конкурентних відеоігор. Технологія моделювання допомагає забезпечити середовище проб і помилок для глибокого навчання з підкріпленням, яке є масштабованим і де помилки не завдадуть реальної шкоди.

3.1 Навчання з підкріпленням

Основними математичними поняттями, на яких базується навчання з підкріпленням є стан та множина станів, дія та множина дій, стратегія та множина стратегій, функція користі та функція користі пари стан-дія. Роботу методів навчання з підкріпленням теоретично обґрунтовано для тих випадків, коли модель системи, що навчається можливо подати у вигляді Марковського

процесу прийняття рішень [3]. Для цього потрібне виконання марковської властивості відносно еволюції системи, тобто її наступний стан повинен залежати лише від поточного стану та обраної дії, а не від її історії. Це виражається формулою:

$$P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$$

де s – стан системи, r – винагорода, a – дія.

Для досягнення цієї умови необхідно правильно визначити як описувати стан системи. Також для гарантованої збіжності алгоритмів необхідно повне знання системи агентом, хоча для більшості реальних випадків без повного знання працює і вибірковий метод проб та помилок. У загальному випадку процес навчання можна описати, як на рис. 3.1 [4].

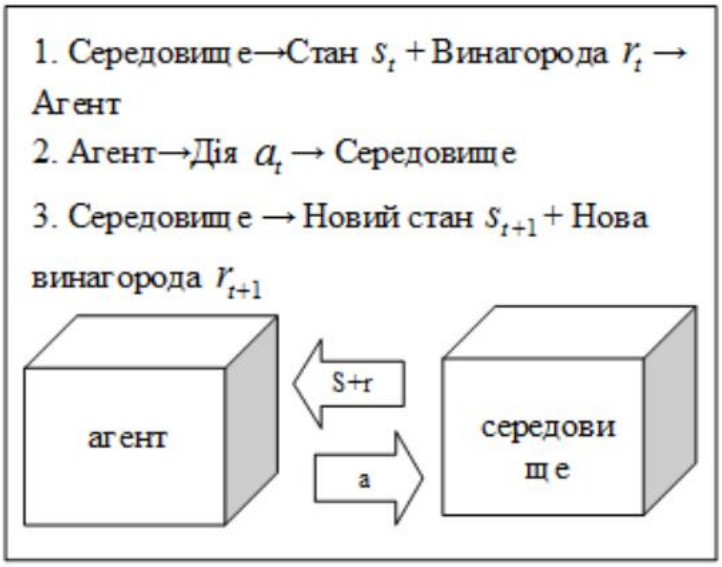

Рисунок 3.1 - Ілюстрація до моделі навчання з підкріпленням: взаємодія агента з середовищем

Спершу середовище, що знаходиться у певному стані s з множини станів S, передає деяку винагороду r агенту, що разом з минулими винагородами впливає на вибір стратегії поведінки π. У відповідь агент виконує деяку дію a з множини дій A, що змінює стан середовища на новий s'. Середовище посилає нову винагороду r' і починається новий крок. Стратегія – це відображення π :

$S \times A \rightarrow [0; 1]$, тобто визначена ймовірність виконати агентом певну дію a, коли середовище знаходиться у певному стані s. Результатом повинна бути максимізація певної сумарної винагороди. Необхідно враховувати, що не завжди вибір дії, що дасть більшу винагороду на наступному кроці, призведе до максимуму усієї винагороди. Зазвичай у якості сумарної винагороди вводять функцію вигляду $R = rt+\gamma\ rtt\ +\ 1+\gamma\ 2\ rt+2+. . .$, де $\gamma \in [0; 1]$ – коефіцієнт, що визначає важливість наступних винагород у порівнянні з тільки що отриманими.

Алгоритм навчання з підкріпленням: визначається функція вартості (V) для вимірювання того, наскільки хорошим є певний стан з точки зору очікуваної сукупної винагороди для агента, який дотримується певної політики. Таким чином, функція оптимального значення (V *) є такою, яка дає максимально досяжне значення для кожного стану в даному просторі станів (набір усіх можливих станів).

Функція Q-значення (Q) показує, наскільки певна дія є виграшною для агента, який дотримується політики, за певного стану. Оптимальна функція Q-значення (Q *) дає максимальну віддачу, досяжну від заданої пари стану дії будь-якою політикою. Однією з ключових властивостей Q* є те, що він повинен задовольняти рівняння оптимальності Беллмана, згідно з яким оптимальне значення Q для даної пари стану дії дорівнює максимальній винагороді, яку агент може отримати за дію в поточному стані + максимальна винагороду зі знижкою, яку він може отримати від будь-якої наступної пари-дії, що випливає з цього. Рівняння виглядає так [11]:

$$Q * (s, a) = E [Rt + 1 + \gamma \max a' Q * (s', a')]$$

Оскільки Рівняння Беллмана допомагає обчислити Q* на кожному кроці, воно дає спосіб визначити оптимальну політику; розраховуємо значення наступної пари-дії, тому оптимальною стратегією є відтворення дій, які максимізують очікуване значення.

Оптимальна політика π*, полягає у вживанні найкращих дій - як визначено Q * - на кожному кроці.

3.1 Алгоритм Deep Q Learning

Компанія DeepMind створила систему штучного інтелекту, яка в більшу частину ігор Atari (Рисунок 3.2) грає краще за людину. Програма навчилася грати, не знаючи правил і не маючи доступу до коду, а просто спостерігаючи за картинкою на екрані.

Побудову та реалізацію Deep Q-learning (DQN) здійснив лондонський підрозділ Google DeepMind. Штучному інтелекту не повідомляли правила гри. Нейромережа сама аналізувала стан і шукала спосіб набрати максимальну кількість очок.

Універсальна система яка може самостійно навчатися може знайти застосування в автономних автомобілях і інших проектах, де потрібно аналізувати стан навколишніх об'єктів і приймати рішення.

Ігри Atari запускалися через емулятор. На кожному кроці агент вибирав одну із доступних дій, який визначався кнопками геймпада. Сам агент не отримував стану емулятора. Замість цього він отримував зображення з емулятора. В якості нагороди використовувався рахунок із гри.

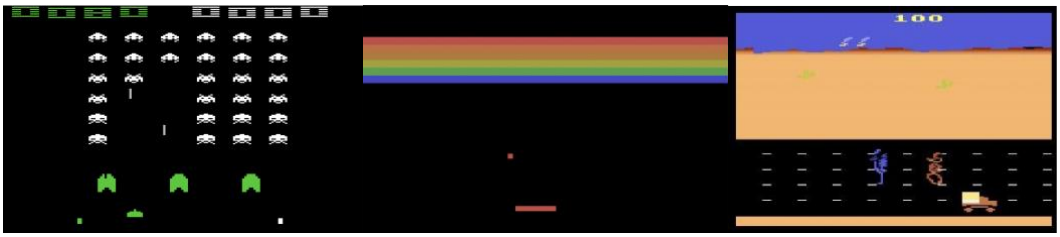

Рисунок 3.2 – Приклади ігор Atari. Зліва направо: Space Invaders, Breakout, Road Runner

Задачу формалізували під марковський процес прийняття рішень, дозволяючи використовувати стандартні методики навчання із підкріпленням.

Алгоритм deep Q-learning використовує згорткову нейронну мережу (Рисунок 3.3) як функцію апроксимації числа Q. Він, після деяких трансформацій, приймає картинку із емулятора на вхід. Алгоритм трансформує вхідні дані за допомогою пари шарів нелінійних функцій. На виході отримуємо 18-мірний вектор. Цей вектор виражає поточний стан і кожну із можливих дій. Дія вибирається серед найвищих Q.

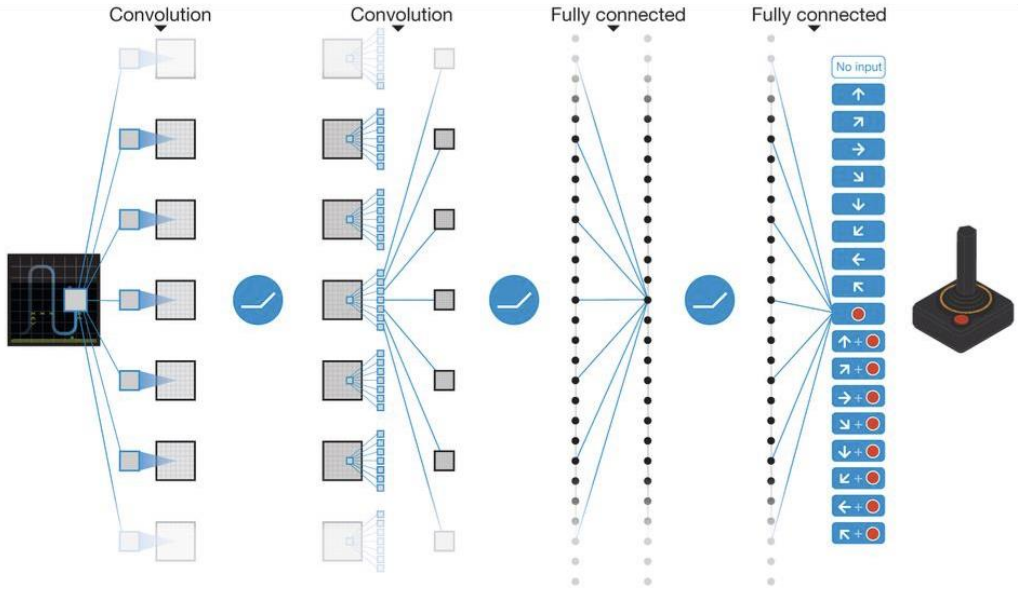

Рисунок 3.3 - Схематичні ілюстрація згорткової нейронної мережі[13]

Спочатку, при тренуванні, цінність кожної дії невідома. І при першому проході рандомно вибрані дії можуть покращити свою цінність Якщо їх цінність буде більша за інші, неперевірені дії то, при жадібній стратегії, будуть вибиратися тільки вони, навіть якщо це не кращий вибір.

Рішенням цієї проблеми є використання стратегії epsilon-greedy під час тренування. В агента є незначний шанс ε вибрати випадкову дію замість вибору дії з кращим Q. Із-за цього кожна дія буде вибиратися не із-за Q чисел ініціалізації а із-за накоплення певного досвіду [12]. Нейронна мережа отримує стани з навколишнього середовища як вхідні дані, а потім виробляє

розрахункові значення Q для кожної дії, яку агент може вибрати в цих станах. Q * має відповідати рівнянню Беллмана тому ми візьмемо цільові значення справа -ручна його частина. Потім ми порівняємо результати роботи моделі з цільовими значеннями та обчислимо втрати.

Далі, використовуючи алгоритм зворотного зв’язку та стохастичний градієнтний спуск, ми отримаємо від нейронної мережі оновлення своїх значень, щоб мінімізувати помилку.

Atari 2600 використовувалася як зовнішній пристрій для телевізорів. Вона генерувала 60 кадрів в секунду. Але алгоритму DQN не було потрібно стільки багато кадрів. Навпаки обробка стількох кадрів сповільнювала тренування алгоритму. Тому експериментальним шляхом було вирішено використовувати тільки кожен 4 кадр для всіх ігор крім space invaders, де такий пропуск кадрів робив лазер невидимим. Тому там було вирішено використовувати кожен 3 кадр що є єдиною різницею між настройками для всіх ігор.[12]

3.2 Висновки до третього розділу

В цьому розділі була розглянута архітектура та алгоритм навчання з підкріпленням. Проаналізована поведінка агента в невизначеному середовищі. Наведені основні математичні поняття що описують стратегію та винагороду агента.

Були проаналізовані сучасні приклади застосування машинного навчання а саме розробка компанії DeepMind - Q-Learning, розглянута Deep Q-Learning у застосуванні до ігор Atari 2600. DQL показав значні результати: у 22 із 49 ігор обігнала людські результати. Результати проти спеціалізованих алгоритмів були навіть кращі – результати були більші в 43 із 49 ігор.

Враховуючи, що на DQN надходила тільки картинка з екрану та рахунок, а нейронна мережа не мала ніяких змін в порівнянні з нейронними мережами для інших ігор, можна припустити що DQN можна використовувати в проектах, де потрібен аналіз навколишнього середовища, а також в середовищах динамічної зміни таких як в грі MineCraft.

РОЗДІЛ 4. МЕТОДИКА НАВЧАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Основна задача комп’ютерного зору полягає в тому, щоб надати машинам можливість бачити світ так, як це роблять люди, сприймати його подібним чином і навіть використовувати знання для безлічі завдань, таких як розпізнавання зображень та відео, аналіз зображень та класифікація, відтворення засобів масової інформації, рекомендаційні системи, обробка природних мов і т.д. Просування в області комп’ютерного зору було удосконалено, насамперед, за допомогою згорткової нейронної мережі (Convolutional Neural Network, CNN) [20].

4.1 Архітектура Convolutional Neural Network

Нейронна мережа - це послідовність різноманітних нейронів, з'єднаних між собою синапсами. Згорткова нейронна мережа або коротко CNN - це спеціалізований тип нейронної мережі, призначений для роботи з двовимірними даними зображення, дані представлені у двовимірному та тривимірному просторі. Центральним для CNN є згортковий шар, який дає мережі назву. У контексті згорткової нейронної мережі згортка - це лінійна операція, яка передбачає множення набору ваг із входом, подібно до традиційної нейронної мережі[21]. В архітектуру мережі закладені апріорні знання з предметної області комп’ютерного зору: піксель зображення сильніше пов’язаний з сусіднім (локальна кореляція) і об’єкт на зображенні може зустрітися в будь якій частині зображення.

На Рисунок 4.1 зображена архітектура згорткової нейронної мережі[22]. CNN складається з різних видів шарів: згорткові шари (convolutional),

субдискритезуючі шари (Pooling or sub-sampling layer) і шари звичайної нейронної мережі – персептрона.

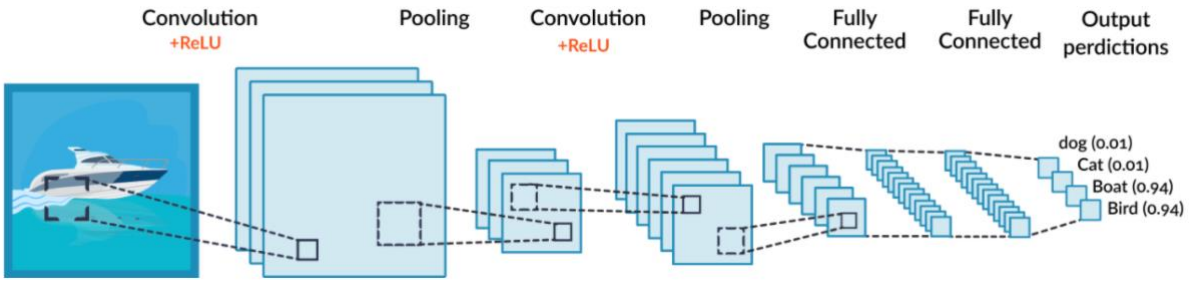

Рисунок 4.1 - Архітектура CNN

На вхід нейронна мережа отримує картинку, в нашому випадку це знімок з гри MineCraft, який розділяється на 3 канали RGB тобто червоний, зелений, та синій (Рисунок 4.2).

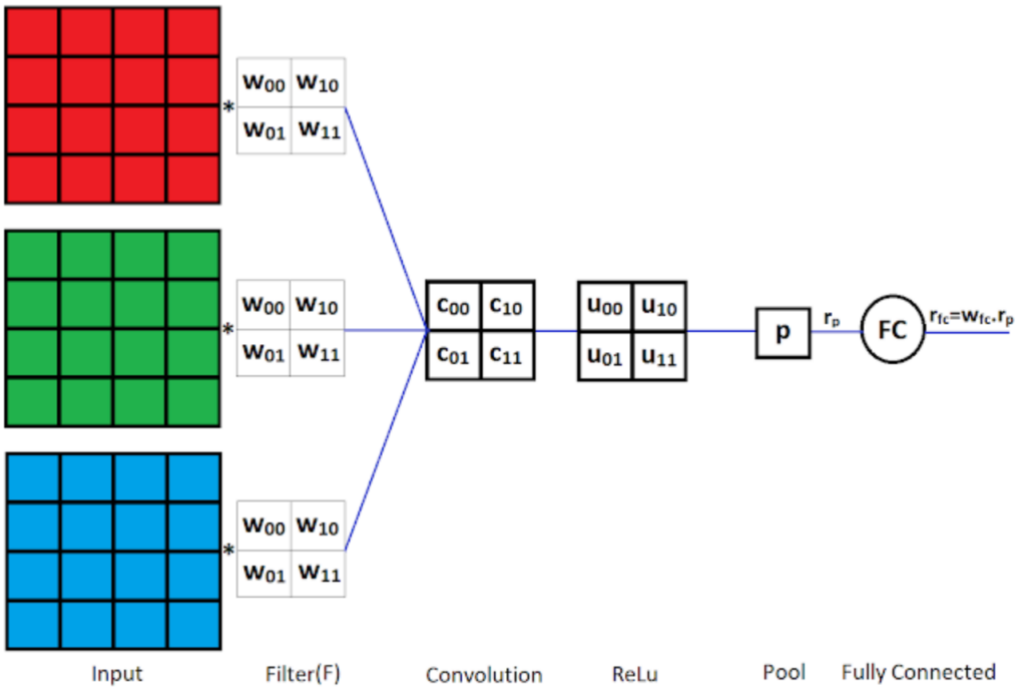

Рисунок 4.2 – CNN із кольоровими входами та кодованими спектрами

У вхідних даних [32×32×3] міститься вихідна інформація про зображення (в даному випадку 32 — ширина, 32 — висота, 3 — кольорові канали R, G, B). Далі розглянемо архітектуру згорткової нейронної мережі детально.

4.1.1 Шар згортки (convolutional layer)

Шар згортки є ключовим компонентом CNN і, як правило, становить принаймні їх перший шар (Рисунок 4.3) [24]. Згортковий шар це набір різного роду карт (матриць), у кожній з цих карт є в наявності синаптичне ядро, або як його ще називають фільтром або скануючим ядром. Кількість карт прямо корелює з якістю розпізнавання об’єктів на зображенні, чим більша кількість карт тим якість вища. Проте водночас збільшується обчислювальна потужність яка може стати обмежуючою силою та вирішення задачі. Наразі найпопулярніше співвідношення один до двох, тобто кожна матриця попереднього шару пов'язана з двома картами згортками.

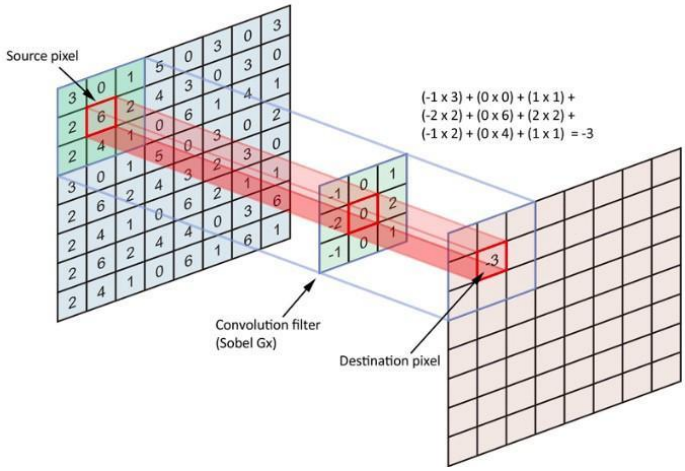

Рисунок 4.3 - Шар згортки (convolutional layer)

Формулою згортковий шар можна описати наступним чином[16]:

$$x_j^l = f(\sum_i x_i^{l-1} * k_j^l + b_j^l)$$

$xj\ l$ – карта ознак j (вихід шару l),

f()– функція активації,

bj – коефіцієнт здвигу для карти ознак j,

kj – ядро згортки номер j,

$xi\ l{-}1$ – карта ознак попереднього шару.

За рахунок крайових ефектів розмір вихідної матриці зменшується. Вихідний сигнал j,k-ого нейрона, в рамках однієї карти ознак обчислюється за формулою так званого ядра або фільтра згортки:

$$\sigma(b + \sum_{l} \sum_{h} w_{l,h} a_j + l, k + h)$$

На рисунку 4.4 ядро являє собою фільтр або вікно, яке ковзає по всій області попередньої карти і знаходить якісь ознаки об'єктів, та будує так звану карту активації або карту характеристик. Наприклад, навчаючи модель на знімках гри MineCraft, то одне з ядер після навчання буде давати найбільший сигнал в області будинку, дерева чи скарбу. Якщо розмір ядра занадто малий, то воно не може виділити будь-які ознаки, таким чином необхідно правильно підібрати розмір ядра. Також розмір ядра вибирається таким чином, щоб розмір карт згорткового шару був парний, це дозволяє не втрачати інформацію при зменшенні розмірності в підвибірковому шарі.

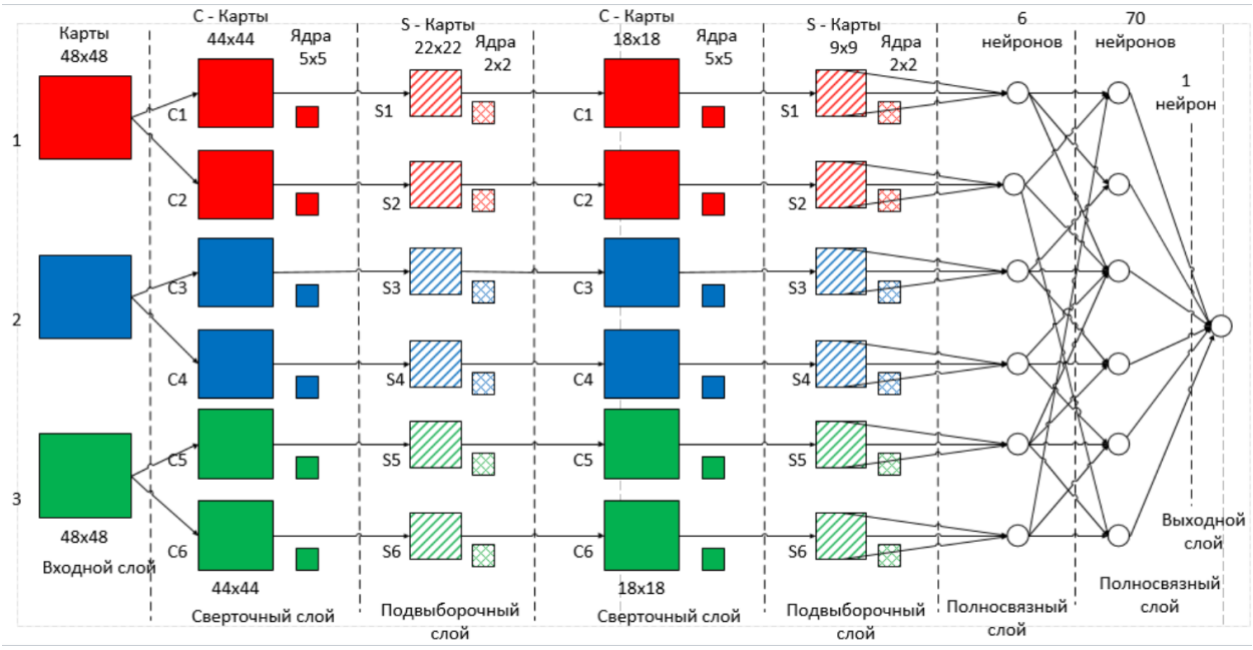

Рисунок 4.4 – Зв’язок згорткового шару з попереднім

Для розпізнання однієї ознаки використовується лише одна карта. Для кожної ознаки потрібно створити свою карту з власним ядром. Архітектурно, внутрішні шари складаються з десятки та сотень карт ознак. Початкове зображення подається на вхідний шар. У першому шарі підвибірки кожна карта ознак здійснює пошук певної, закріпленої тільки за даною карткою, ознаки. Досягається це за рахунок використання загальної для всієї карти ознак матриці ваг і особливої організації локального рецептивного поля для кожного нейрона такої карти. Кожен нейрон карти ознак отримує вхідні дані від прямокутної області розміру n × m вхідного зображення. Така область досить мала і безліч таких областей на вхідному зображенні перетинаються і накладаються за принципом черепиці. Суміжні нейрони карти ознак отримують в якості вхідного впливу суміжної прямокутної області, причому вагові коефіцієнти для всіх нейронів карти ознак будуть однаковими. Для простоти викладу будемо називати область, котра формує локальне рецептивне поле нейрона шару підвибірки, вікном. Відповідно, площу вікна вважати - кількістю нейронів в такій області. Таким чином, можна говорити про те, що карта ознак в цілому здійснює операцію пошуку ознаки у вхідних даних. Інші карти ознак мають інший набір вагових коефіцієнтів і, відповідно, здійснюють пошук інших ознак у вхідних даних. Після виконання згортки мережа втрачає частину інформації про точне положення знайденої ознаки, але зберігає інформацію щодо взаємного розташування різних ознак.

Ядро являє собою систему поділюваних ваг або синапсів, це одна з головних особливостей згорктової нейронної мережі. У CNN загальні ваги дозволяють скоротити число зв’язків і знаходити одну ознаку по всій області зображення в той час як у звичайній багатошаровій мережі дуже багато зв’язків між нейронами. Ядра ваг (фільтрів) позначають ваги шару згортки. Вони ініціалізуються, а потім оновлюються шляхом зворотного розповсюдження градієнта[25].

Переваги згорткової нейронної мережі в тому що вона здатна визначити всі елементи зображення. Наприклад, якщо задача відрізнити два об’єкти один від одного, тобто задача класифікації, то мережа легко визначить ознаки кожного об’єкту.

4.1.2 Шар об’єднання (pooling layer)

Цей тип шарів часто розміщується між двома шарами згортки: він отримує кілька вхідних карт у якості вхідних даних і застосовується до кожного з них операції об'єднання (Рисунок 4.5). Шар субдискритезуючий або пулінгу використовується для зменшення розмірів карт. Таким чином, це зменшує кількість параметрів для вивчення та обсяг обчислень. Рівень об’єднання узагальнює об’єкти, присутні в області карти об’єктів, породженої шаром згортки. Отже, подальші операції виконуються на узагальнених елементах замість точно позиціонованих об’єктів, породжених шаром згортки. Це робить модель більш надійною до варіацій положення елементів на вхідному зображенні.

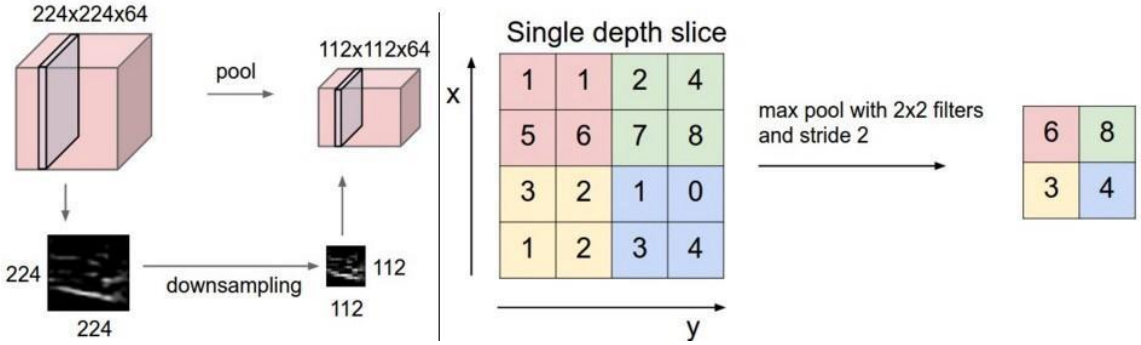

Рисунок 4.5 - Шар об’єднання (Pooling layer)

Шар об’єднання можна описати за наступною формулою[17]:

$$x^l = f(a^l \times subsample(x^{l-1}) + b^l)$$

- xl – вихід шару l,
- f()– функція активації,
- a,b – коефіцієнти,
- subsample – операція вибірки локальних максимальних значень

Як і згорткові шари, оператори об'єднання складаються з вікна фіксованої форми, яке ковзає по всіх регіонах вводу відповідно до його кроку, обчислюючи один вихід для кожного місця, пройденого вікном фіксованої форми (іноді відомий як вікно об'єднання). Однак, на відміну від обчислювальної кореляції вхідних даних і ядер у згортковому рівні, рівень об'єднання не містить параметрів (ядра немає). Натомість оператори об’єднання є детермінованими, як правило, обчислюють або максимальне, або середнє значення елементів у вікні об’єднання. Ці операції називаються Max Pooling та Average Pooling відповідно. Наприклад розглянемо Max Pooling з формою вікна 2×2. Затінені ділянки є першим вихідним елементом, а також вхідними тензорними елементами, що використовуються для обчислення вихідних даних: max (0,1,3,4) = 4. (Рисунок 4.6).

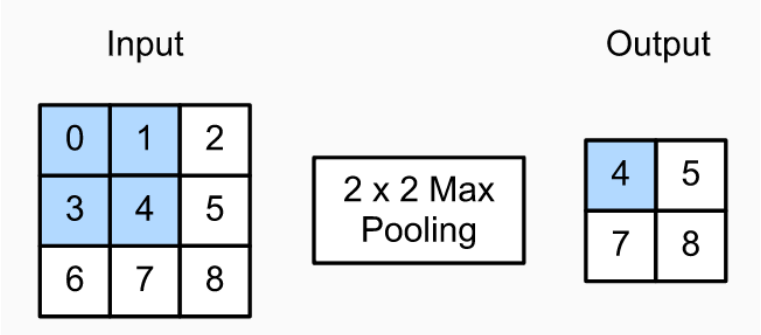

Рисунок 4.6 – Max Pooling

4.1.3 Активаційна функція ReLU

ReLU (Rectified Linear Units) позначає реальну нелінійну функцію, визначену ReLU (x) = max (0, x) (Рисунок 4.7) [27].

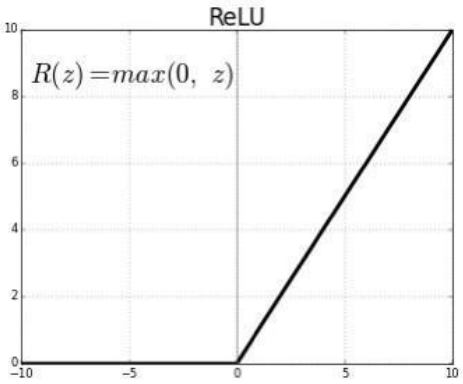

Рисунок 4.7 - Активаційна функція Relu

Тому корекційний шар ReLU замінює всі негативні значення, отримані як входи, нулями. Він відіграє роль функції активації.

4.1.4 Повно зв'язаний шар (Fully-connected layer)

Fully-connected layer завжди є останнім шаром нейронної мережі, згорнувшись чи ні, тому це не є чіткою характеристикою CNN (Рисунок 4.8) [28].

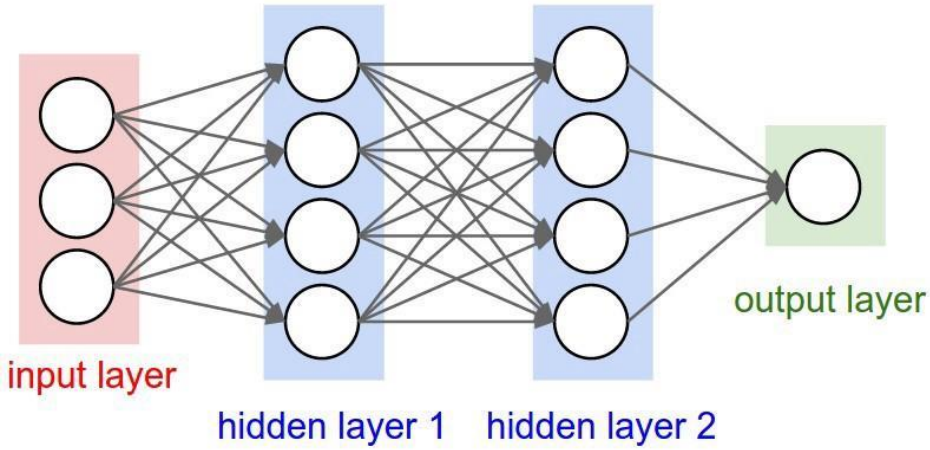

Рисунок 4.8 - Повністю з'єднаний шар (Fully-connected layer)

Цей тип шару приймає вхідний вектор і виробляє новий вихідний вектор. Для цього застосовується лінійна комбінація та необов'язкова функція активації до значень, отриманих як вхід.

Останній повністю пов'язаний шар класифікує вхідне зображення мережі: він повертає вектор розміром N, де N - кількість класів у нашій проблемі класифікації зображень. Кожен елемент вектора вказує на ймовірність того, що вхідне зображення належить до класу[29].

Кожне значення вхідного масиву «голосує» на користь класу. Не всі голоси мають однакове значення: шар надає їм ваги, які залежать від елемента таблиці та класу.

Щоб обчислити ймовірності, повністю пов'язаний шар тому множить кожен вхідний елемент на вагу, підсумовує його, а потім застосовує функцію активації (логістична, якщо N = 2, softmax якщо N> 2).

Ця обробка означає множення вхідного вектора на матрицю, що містить ваги. Те, що кожне вхідне значення пов'язане з усіма вихідними значеннями, пояснює пов'язаний термін.

Згорткова нейронна мережа вивчає значення ваг так само, як і вивчає фільтри шару згортки: під час тренувального етапу шляхом зворотного розповсюдження градієнта[30].

Повністю пов'язаний шар визначає зв'язок між властивостями зображення і класу. Фактично, оскільки вхідний масив є результатом більш раннього шару, він відповідає карті активації функції: високі значення вказують місце розташування цієї функції на зображенні.

4.2 Навчання згорткової нейронної мережі

Метод стохастичного градієнтного спуску – це метод навчання нейронної мережи. Це ітеративний градієнтний алгоритм, який використовується з метою мінімізації помилки та отримання бажаного виходу.

Основна ідея цього методу полягає в поширенні сигналів помилки від виходів мережі до її входів, в напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи[19].

Вхідними даними для нейронної мережі являється двадцяти канальне зображення 144 на 144 пікселів. Вихідне зображення має дещо менші розміри (на постійну ширину границі) ніж вхідне, за рахунок згортки. Далі, функція soft-max попіксельно розраховується по карті, разом з функцією кросентропії. Кросентропія в кожній точці розраховується за наступною формою[21]

$$E = \sum_{x \in \Omega} \omega(x) \log(p_{l(x)}(x))$$

Потім розраховуються карта вагових коефіцієнтів за наступною формулою[21]:

$$\omega(x) = \omega_c(x) + \omega_0 \times \exp(-\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}$$

ωc - карта вагів для балансування частот класів,

$d1$ - відстань до границі найближчої соти,

$d2$ - відстань до границі другої найближчої соти.

Для того, щоб провести ітерацію навчання, необхідно зрозуміти як через них виражається значення градієнтів функції помилки ваг. Через функцію отримання максимуму помилка проходить без змін, шар дискретизації не виконує навчання. Однак, прохід через граф розраховані градієнти розрідженими, адже з усіх елементів вікна субдескритезації $zi,j\ l$, приватна похідна $\partial E\ \partial xi,j\ l$+1 відноситься тільки до одного, максимального, а всі інші

отримають нульовий градієнт, на цьому процес навчання можна вважати завершеним.

Далі, необхідно виконати процедуру пропускання через не лінійність:

$$\frac{\partial E}{\partial y_{i,j}^{l}} = \frac{\partial E}{\partial z_{i,j}^{l}} \frac{\partial z_{i,j}^{l}}{\partial y_{i,j}^{l}} = \frac{\partial E}{\partial z_{i,j}^{l}} h`(y_{i,j}^{l})$$

На даному етапі відомо $\partial E\ \partial zi,j\ l$, а $h`(yi,j\ l)$, можна отримати шляхом розрахунків. В свою чергу, на згортковому рівні з'являються ваги, котрі потрібно буде навчати. Певна складність тут полягає у тому, що всі ваги діляться і кожен приймає участь у всіх результатах, таким чином отримана сума досить значна.

Для повноти розуміння залишилось лише пропустити градієнти через попередній шар[24].

$$\frac{\partial E}{\partial x_{i,j}^{l}} = \sum_{a}\sum_{b} \frac{\partial E}{\partial y_{i-a,j-a}^{l}} \frac{\partial y_{i-a,j-b}^{l}}{\partial x_{i,j}^{l}} = \sum_{i}\sum_{j} \frac{\partial E}{\partial y_{i-a,j-b}^{l}} w_{a,b}$$

Зворотній прохід для згортки виявився дуже схожим на згортку з тими ж вагами wa,b, тільки замість $i + a$ та$j + b$ тепер $i - a$ та $j - b$. В тому випадку, коли зображення доповнюються нулями, розмірності зберігаються, зворотній обхід можна вважати такою ж згорткою, як і прямий прохід, тільки розвернутими осями.

При навчанні мережі на зображеннях гри MineCraft було використано 1000 зображень з розміром 144x144 пікселів, та отримано близько 100 тисяч фрагментів, котрі не перетинаються, чого достатньо для навчання мережі. При такій кількості даних варто застосувати метод для запобігання перенавчання

(overfitting), тому було використано метод перехресної перевірки (cross-validation) та метод виключення (dropout).

4.3 Метрики оцінки якості роботи класифікатора

Для аналізу оцінки ефективності роботи класифікаторів було обрано декілька метрик для порівняння: Accuracy, Precision, Recall, F1 score та ROC AUC score [28].

Оцінка якості роботи класифікаторів базується на понятті Матриці Помилок (Confusion Matrix), що для задачі бінарної класифікації має наступний вигляд.

True Positive – випадок, коли реальне значення дорівнювало 1

True Negative – випадок, коли реальне значення дорівнювало 0

False Positive – випадок, коли реальне значення дорівнювало 0, а прогнозоване – 1.

False Negative,- випадок, коли реальне значення дорівнювало 1, а прогнозоване – 0.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Accuracy – ефективна метрика якості, коли маємо справу із збалансованими класами. Однак, якщо кількість об’єктів одного класу значно перевищує інший, то точність покаже хороші результати, що насправді будуть неправдивими.

$$Precision = \frac{TP}{TP+FP}$$

Точність можна інтерпретувати як долю об'єктів, названих класифікатором Positive і при цьому дійсно об'єкти являються Positive.

$$Recall = \frac{TP}{TP+FN}$$

Повнота показує, яку долю об'єктів позитивного класу з усіх об'єктів позитивного класу алгоритму класифікував вірно[30].

$$\text{F1 score} = \beta^2 * \frac{Precision * Recall}{\beta^2 Precision + Recall}$$

При оптимізації гіперпараметрів використовується одна метрика, поліпшення якої ми і очікуємо побачити на тестовій вибірці. Існує кілька різних способів об'єднати precision і recall в агрегований критерій якості. F-міра - середнє гармонійне precision і recall.

При конвертації вихідних даних в бінарну мітку, ми повинні вибрати будь-якої поріг, при якому 0 стає 1. Природним і близьким здається поріг, рівний 0.5, але він не завжди виявляється оптимальним, наприклад, при відсутності балансу класів.

Одним із способів оцінити модель, не прив'язуючись до конкретного порогу, є AUC-ROC (або ROC AUC) - площа (Area Under Curve) під кривою помилок (Receiver Operating Characteristic curve). Дана крива представляє із себе лінію від (0,0) до (1,1) в координатах True Positive Rate (TPR) і False Positive Rate (FPR) (Рисунок 4.9) [37]

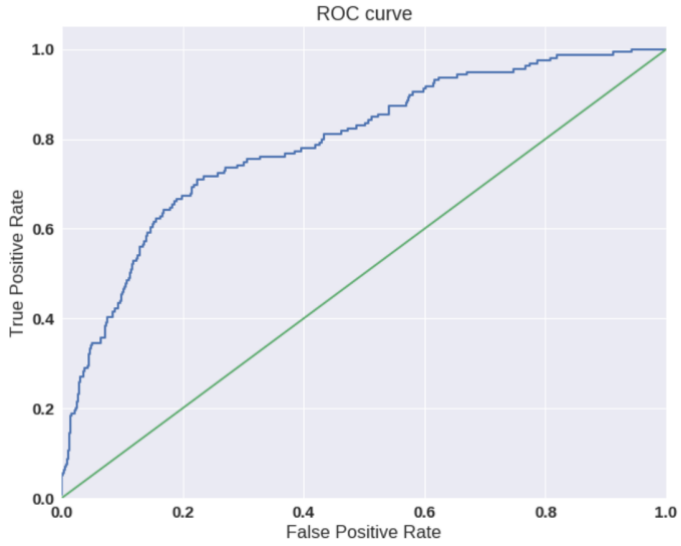

Рисунок 4.9 - ROC-крива

Критерій AUC-ROC стійкий до незбалансованим класів і може бути інтерпретований як ймовірність того, що випадково обраний positive об'єкт буде проранжованим класифікатором вище (матиме більш високу ймовірність бути positive), ніж випадково обраний negative об'єкт.

4.4 Архітектура DQN для класифікації зображень в грі Minecraft

Deep Q Learninng це метод навчання з підкріпленням, який об’днує Q-Learning функцію з глибинними нейронними мережами, щоб дозволити агенту навчатись в складних середовищах що змінюються, таких як відеоігри або робототехніка (Рисунок 4.10).

Для побудови згорткові нейронної мережі та симуляції гри були використані Python3 та бібліотека Tensorflow а також Google Colab Notebook аби запускати експерименти на GPU.

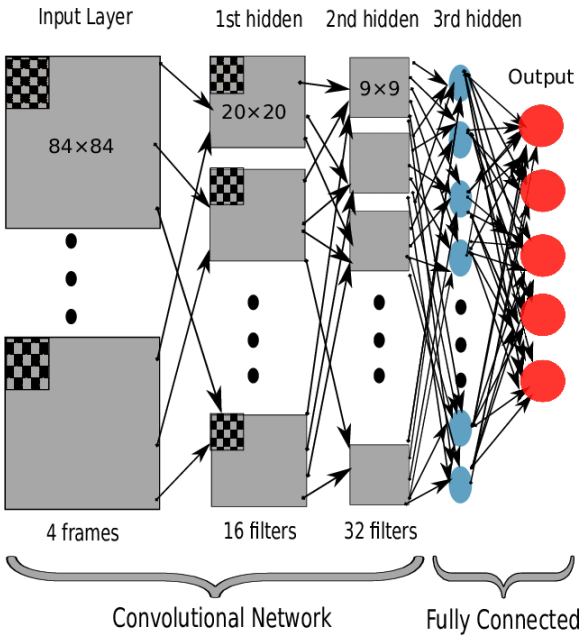

Рисунок 4.10 – Архітектура DQN

Згорткова нейронна мережа представлена з пікселями на вході з кількістю нейронів на виході рівною кількості можливих дій агента. Більш точно - на вхід мережі видаються пікселі кількох минулих кадрів емулятора (4-х кадрів), тобто мережа може відстежити рухомі об'єкти. На кожному нейроні останнього шару мережі - Q (s, a) для потрібного a.

При першому кроці мережа з деякими вагами, далі в залежності від вихідних даних агент здійснює крок, тобто ми дізналися новий перехід з s в s' в результаті дії a і чи була нагорода (reward), в грі MineCraft чи знайшов агент скарб. Отримуємо набір (s, s ', a, r), виберемо з них кілька випадкових, і скористаємося рівнянням Белмана, щоб підкоригувати ваги мережі, аби вони передбачали максимум з того, що на наступному кроці передбачали старі ваги. В пам'яті алгоритму запам'ятовується всі переходи агентта, які модель бачила і тренуємо нейронну мережу, яка по картинці пророкує куди нас приведе наступна дія. У самій нейромережі пам'яті про стани немає, проте всі кроки збереглись в натренованих вагах. Для побудови більш стійкої моделі використовується e-greedy policy, тобто на кожному кроці з ймовірністю e буде

зроблено випадкова дія, а з ймовірністю (1-e) передбачене поточним розрахунком Q-функції, з часом e, зменшується.

4.5 Висновки до четвертого розділу

В даному розділі було розглянуто основні особливості згорткової нейронної мережі, навчання з підкріпленням та комбінацію DQN, а саме компонентів з яких вона складається та алгоритм навчання.

Ключові моменти:

− у якості тренувального набору було обрано набір даних з гри MineCraft, який складається з 60 млн;

− для дослідження обрано десять класів (building, structures, road, cars, dog, cat, bucket, block, Minecraft kelp and diamond як ціль);

− оптимальним вибором для виконання завдання виявилась нейронна мережа архітектури DQN;

− у якості міри оцінювання ефективності роботи нейронної мережі обрано Precision, Recall, F1 score, ROC AUC.

РОЗДІЛ 5. ОЦІНЮВАННЯ ЯКОСТІ РОБОТИ КЛАСИФІКАТОРІВ ДЛЯ ОЦІНКИ ПОЛІТИКИ АГЕНТА

Набір даних складає 60 млн картинок з гри MineCraft, який був розділений на навчальну та тестову підвибірку (60% та 40% відповідно), для тренування агента важливий порядок зміни кадрів тому розбиття відбулось по завершеній грі.

Для експериментів та симуляції гри були використані Google Colab Notebook та Google GPU. Було побудовано та проаналізовано чотири класифікатора: метод опорних векторів, дерева рішень, наївний баєйсьвський класифікатор та згорткова нейронна мережа.

Середовищем для класифікації зображень та використання методу Q-learning, була обрана комп'ютерна гра MineCraft з розробленими правилами навчання. (Рисунок 5.1). Мета гри полягає в тому щоб гравець знайшов діамант серед усіх об'єктів. Гравець може рухатись в будь якому напрямку, мандрувати лабіринтами та зустрічати на своєму шляху різні об'єкти. У разі знаходження скарбу (діаманту), гра вважається успішно завершена.. Бали (scores) набираються з кожним успішним знаходженням скарбу проте відміняються за кожен додатковий крок.

Модель навчання полягає у тому що агент обирає чи виконати крок у середовищі та в якому напрямку. У якості стану використовувалась відстань від агента до ближньої перешкоди тобто об'єкта на координатній сітці (x, y), а також його швидкість (v) за віссю ординат. Q-матриця представляла собою словник з ключем у вигляді (x, y, v), елементами якого є масиви з вагами можливих дій агента. За кожну правильну дію він отримував винагороду +1000 балів за успішне знаходження скарбу, а за додатковий крок— покарання -10. Перед здійсненням кроку, модель класифікації передавала ймовірність кожного класу об'єкта в полі зору агента. У моделі згорткової нейронної мережі, у вагах, були натренеровані коефіцієнти які відповідали за маршрут

агента, тобто додатково обирався напрямок руху де ймовірність знайти скарб вищий.

Прийняття рішення агентом відбувається 15 разів у хвилину, оскільки гра розраховує 60 кадрів у хвилину а вхідними даними для згорктвої нейронної мережі є кожен 4 кадр. Мета агента — набрати найбільшу кількість балів у заданому штучному середовищі.

Стратегія навчання.

У ході розробки агента застосовано жадібну стратегію Q-навчання, яка полягає в тому, що вибрано дію з найкращим Q. Причина її вибору полягає в тому, що без використання жадібної стратегії у більшості випадків агент не проходив навіть першу пару перешкод.

Експерименти. Побудувати моделі класифікації об’єктів та знайти оптимальні параметри методу Q-навчання, завдяки яким агент буде отримувати високі бали у середовищі. Для знаходження значень параметрів агент навчався у середовищі впродовж 100 епох по 5000 ітерацій, потім параметри методу $\gamma \in [0,0;\ 0,2;\ 0,5;\ 0,8;\ 1,0]$ та $\alpha \in [0,0;\ 0,2;\ 0,5;\ 0,7;\ 1,0]$ змінювалися, а процес повторювався. Результати навчання агента та моделей класифікацій описані нижче.

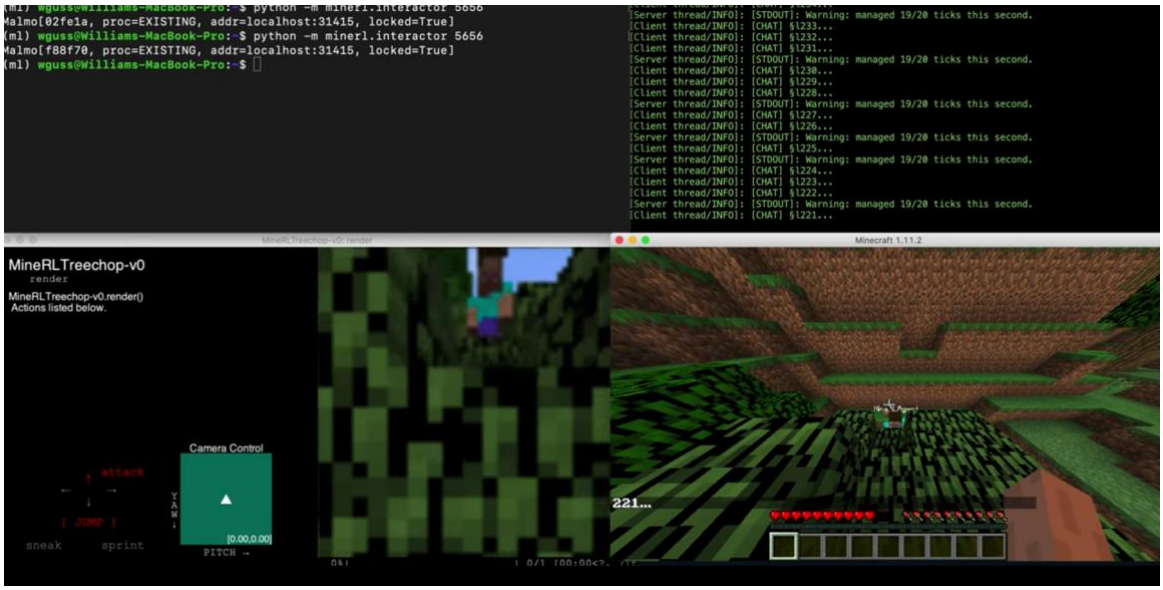

Рисунок 5.1 – симуляція агента за допомогою натренованої CNN

5.1 Результати застосування класифікаторів

5.1.1 Результат застосування методу опорних векторів

Для побудови моделі опорних векторів були використанні модулі бібліотек: sklearn.grid_search та GridSearchCV для налаштовує параметрів, а також крос-валідацію для аналізу результатів:

- mean: 0.73267, std: 0.02558, params: {'C': 0.001, 'kernel': 'linear'},
- mean: 0.48185, std: 0.22123, params: {'C': 0.001, 'kernel': 'rbf'},
- mean: 0.76898, std: 0.04200, params: {'C': 0.01, 'kernel': 'linear'},
- mean: 0.48185, std: 0.22123, params: {'C': 0.01, 'kernel': 'rbf'},
- mean: 0.80858, std: 0.05245, params: {'C': 0.1, 'kernel': 'linear'},

Отже, найбільш доцільним є використання значення параметра *C*, що дорівнює 0.1 та лінійного ядра (табл 5.1).

Таблиця 5.1 – Результати роботи методу опорних векторів

	accuracy	**precision**	**recall**	**specificity**	**f1 score**	**roc auc score**
feature selection (decision trees)	0.79	0.59	0.56	0.81	0.54	0.56
feature selection (rfecv)	0.78	0.56	0.54	0.82	0.54	0.56
feature selection (mutual information)	0.77	0.57	0.56	0.81	0.52	0.56
feature selection (svm)	0.73	0.52	0.54	0.79	0.50	0.56
feature selection (pca)	0.80	0.60	0.58	0.82	0.60	0.58

При побудові моделі методу опорних векторів було створено 5 різних налаштувань параметрів для визначення найкращого варіанту. За всіма

показниками ефективності варіант з найвищими показниками стала feature selection (pca) модель: F1 score 0.60 та Recall 0.58.

5.1.2. Результат застосування методу дерева рішень

Класифікатор дерев рішень був побудований за допомогою бібліотеки XGBoost. Результати класифікації наведені в табл 5.2.

Таблиця 5.2 – Результати роботи дерев рішень

	accuracy	**precision**	**recall**	**specificity**	**f1 score**	**roc auc score**
feature selection (max_depth 5, min_child_weight = 1)	0.77	0.74	0.73	0.79	0.74	0.75
feature selection (max_depth 12, min_child_weight = 0.8)	0.78	0.75	0.76	0.78	0.76	0.77
feature selection (max_depth 12, min_child_weight = 8, eta – 0.4)	0.82	0.78	0.83	0.79	0.80	0.82
feature selection (max_depth 12, min_child_weight = 8, eta – 0.4, gamma – 0.6)	0.78	0.75	0.74	0.79	0.75	0.76

При побудови дерев рішень були проведені експерименти з налаштуванням параметрів:

- eta [за замовчуванням = 0,3] – робить модель більш надійною, зменшуючи ваги на кожному кроці
- min_child_weight [за замовчуванням = 1] - визначає мінімальну суму ваг усіх спостережень. Більш високі значення заважають моделі вивчати відносини, які можуть бути дуже специфічними для вибірки Minecraft.
- max_depth [за замовчуванням = 6] - Максимальна глибина дерева

- gamma [за замовчуванням = 0] – вибірка розбивається лише тоді, коли отриманий результат дає позитивне зменшення функції втрат.

Найвищі показники в моделі дерева рішень з налаштуваннями feature selection (max_depth 12, min_child_weight = 8, eta – 0.4): F1 score 0.80 та Recall 0.83.

5.1.3. Результат застосування методу наївного Баєйсівського класифікатора

Результати класифікації за допомогою моделі NB наведені в табл 5.3

Таблиця 5.3 – Результати роботи наївного байєсівського класифікатора

	accuracy	**precision**	**recall**	**specificity**	**f1 score**	**roc auc score**
rescaled (minmax)	0.67	0.66	0.61	0.73	0.62	0.64
rescaled (stand)	0.66	0.65	0.61	0.72	0.61	0.62
feature selection (decision trees)	0.67	0.65	0.60	0.71	0.61	0.64
feature selection (rfecv/ f- classif/ mutual-info)	0.56	0.54	0.53	0.53	0.51	0.56

В експериментах з Наївним Баєйсівським класифікатором було створено 4 моделі з різними параметрами, максимальні результати у моделі rescaled (minmax): F1 score 0.62 та Recall 0.61, проте модель rescaled (standart): показала ідентичні результати Recall 0.61.

5.1.4 Результат застосування згорткової нейронної мережі

Структура репозиторія для побудови нейронної мережі:

```
├── aicrowd.json      # Submission meta information like your username
├── apt.txt           # Packages to be installed inside docker image
├── data              # The downloaded data, the path to directory is also available as
    `MINERL_DATA_ROOT` env variable
├── requirements.txt  # Python packages to be installed
├── test.py           # IMPORTANT: Your testing/inference phase code, must include
    main() method
├── train             # Your trained model MUST be saved inside this directory, must
    include main() method
├── train.py          # IMPORTANT: Your training phase code
└── utility           # The utility scripts to provide smoother experience to you.
    ├── debug_build.sh
    ├── docker_run.sh
    ├── environ.sh
    ├── evaluation_locally.sh
    ├── parser.py
    ├── train_locally.sh
    └── verify_or_download_data.sh
```

Під час тестування агента значення q друкуються на кожному кроці, а кроки виконуються із затримкою на 0,5 секунди, щоб можна перевірити, як змінюються значення в різних ситуаціях. Доступ до значень q здійснюється через action_value.q_values При перегляді значень q під час тестування агента можна побачити, що всі значення q вищі, коли агент стоїть перед деревом. Результати наведені в таб. 5.4

Таблиця 5.4 – Результати роботи згорткової нейронної мережі

	accuracy	**precision**	**recall**	**specificity**	**f1 score**	**roc auc score**
Number of epoch 100, Dropout 0.5	0.90	0.81	0.78	0.95	0.61	0.78
Number of epoch 100, Dropout 0.5, learning rate 1e-1	0.92	0.81	0.82	0.95	0.81	0.80
Number of epoch 100, Dropout 0.5, learning rate 1e-1, L2 regularization	0.92	0.83	0.87	0.95	0.83	0.82

Найвищі результати в згоркової нейронної мережі з параметрами: Number of epoch 100, Dropout 0.5, learning rate 1e-1, L2 regularization, показники F1 score 0.83 та Recall score 0.87.

5.2 Порівняльний аналіз роботи розглянутих моделей

Якщо взяти до уваги природу обраної практичної задачі та значення «0» та «1» у відповідях класифікаторів, можна прийти до висновку, що коректна класифікація об’єктів першого класу є більш важливою, адже краще зайвий раз зробити додатковий хід ніж пропустити скарб. Отже, при порівнянні моделей між собою і виборі кращої варто надавати перевагу оцінці повноти (recall score), адже саме вона оцінює частку правильно класифікованих об’єктів першого класу та F1 score. А вже в другу чергу звертати увагу на інші характеристики моделей. Найкращі результати кожної із використаних моделей наведені в табл 5.5.

Таблиця 5.5 Найкращі показники кожного з алгоритмів класифікації

Метод	**accuracy**	**precision**	**recall**	**specificity**	**f1 score**	**roc auc score**
Наївний байєсовський класифікатор	0.67	0.66	0.61	0.73	0.62	0.64
Дерева рішень	0.82	0.78	0.83	0.79	0.80	0.82
Метод опорних векторів	0.80	0.60	0.58	0.82	0.60	0.58
Згорткова нейронна мережа (CNN)	0.92	0.83	0.87	0.95	0.83	0.82

На основі таблиці, можна зробити висновок що найкращу оцінку повноти дає метод згорткової нейронної мережі 0.87. F-міра також має найвищий показник по CNN 0.83, проте дерева рішень продемонстрували високі результати 0.80 з показником Recall – 0.83.

5.3 Висновки до п’ятого розділу

В даному розділі була проведена велика кількість випробувань різних типів класифікаторів та перевірка якості їх роботи при вирішенні практичної задачі класифікації зображень гри MineCraft. Для кожного із класифікаторів були визначені оптимальні значення параметрів шляхом застосування перехресної перевірки. Кожна із моделей була протестована як на повному наборі ознак, так і на його піднаборах, що були визначені за допомогою найбільш поширених методів feature selection.

Найкращі результати продемонструвала модель згорткова нейронна мережа CNN DQN – Recall 0.87, F1 0.83.

За результатами роботи класифікаторів було виконано порівняльний аналіз та визначено сильні та слабкі сторони використання кожного з них.

РОЗДІЛ 6. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТ

Deep Reinforcement Learning - це важливий тип машинного навчання, коли агент вчиться, як вести себе в середі, виконуючи дії та зберігая результати. За останні роки ми побачили багато досягнень у цій галузі досліджень. Прикладами можуть служити компанія DeepMind яка продала свій алгоритм DQN за 500 млн доларів, і стала переможцем чемпіонату гри Go с AlphaGo у 2016 році, OpenAI та PPO у 2017 році. В майбутньому Deep Reinforcement Learning буде використовуватись для рішення задач в середовищі що швидко змінюється, навіть зараз DRL використовується для робототехніки в таких компаніях як Boston Dynamics.

6.1 Пошук та аналіз ідей

Голова мета даної роботи – розробити методологію з імплементації навчання з підкріпленням у відеоігрових штучних інтелектах та перевірити її ефективність за допомогою серії тестів у ігровому середовищі. Узагальнення цієї ідеї можна побачити в таблиці 6.1.

Таблиця 6.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Застосування навчання з підкріпленням у відеоігровому штучному інтелекті	Імплементація не тренованих попередньо адаптивних механізмів	Реалізація у грі адаптивних механізмів, що здатні підлаштовуватися під зміну параметрів середовища

Даний проект носить теоретичний характер, тому неможливо підняти питання про конкуренцію, але він може бути порівняний з сучасними широко використовуваними техніками реалізації ігрових ШІ, які можна визначити як «статичні». Окрім цього, для порівняння, будуть взяті ігрові ШІ на основі машинного навчання з попереднім тренуванням, які частіше використовуються у академічній сфері, але також можливі у прикладному застосуванні Порівняння наведено у таблиці 6.2

Таблиця 6.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

Техніко-економічні характеристики ідеї	(Потенційні) товари конкурентів			Слабка сторона	Сильна сторона
	Даний проект	ШІ на основі статичних алгоритмів	ШІ на основі машинного навчання з попереднім трен-ням		
Здатність розвиватися	Присутня	Відсутня	Присутня	-	+
Можливість точного налаштування поведінки	Присутня	Присутня	Відсутня	-	+
Надлюдський рівень ігрової стратегії	Відсутня	Відсутня	Присутня	-	-
Необхідність перенавчання у випадку змін ігрових параметрів	Відсутня	Відсутня	Присутня	-	+

6.2 Технічний аналіз ідеї

Оскільки проект, з технічної точки зору, є шаблоном проектування, то він не потребує ніяких окремих технологій для використовування і може застосовуватися у будь-якому ігровому движку та мові програмування обраними розробником. Технології, у контексті яких використовується проект, можна побачити на таблиці 6.3.

Таблиця 6.3 – Технологічна здійсненність ідеї проекту

Аспект проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Ігрове середовища	Ігровий движок	У відкритому доступі є, наприклад, такі: Unreal Engine, Unity	Вільні для використання
Адаптивний алгоритм	Мови загального програмування, що застосовуються у движках	У відкритому доступі є, наприклад, такі: C#, C++, JS	Вільні для використання

6.3 Аналіз ринкових можливостей стартап проекту

Попередня характеристика потенційного ринку стартап-проекту та потенційних клієнтів стартап-проекту представлені в таблиці Х.4 таблиці Х.5 відповідно. Проводиться визначення ринкових можливостей, які можна використати під час впровадження проекту, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб клієнтів.

Таблиця 6.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	20-30
2	Загальний обсяг продаж, грн/ум.од	1500 грн/ум. од.
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Недискримінаційні якісні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	70%

Таблиця 6.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних груп клієнтів	Вимоги споживачів до товару
1.	Потреба створення адаптивних ігрових механік	1. Великий бізнес 2. Середній бізнес	Потребують реалізації у різних ігрових середовищах реалізації у різних ігрових середовищах	Точність роботи шаблону, можливість гнучкого налаштування ШІ

Фактори загроз та фактори можливостей представлені в таблиці 6.6 та в таблиці 6.7 відповідно.

Таблиця 6.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Крадіжка інтелектуальної власності	Крадіжка ідеї або ключової інтелектуальної інновації	Не є проблемою оскільки шаблон програмного проектування не можливо захистити правом інтелектуальної власності.
2.	Отримання несанкціонованого доступу сторонніми особами	Хакерська атака що може призвести до компроментації даних клієнтів	Залучення спеціалістів з інформаційної безпеки Використання засобів шифрування та резервного копіювання
3.	Відсутність ринку	Відсутність шляху збуту товару внаслідок помилкового орієнтування	Ретельний розгляд проблем потенційних клієнтів Консультації із спеціалістами Продовження роботи над шаблоном з метою вдосконалення
4.	Недостача капіталовкладень	Відсутні кошти до моменту виходу на ринок	Не потребує коштів для виходу на ринок

Таблиця 6.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Отримання інвестицій	Отримання капіталу що необхідний для реалізації продукту	Не потребує капіталу для реалізації
2.	Успішна маркетингова політика	В результаті проведеної маркетингової політики отримана висока зацікавленість користувачів	Продовження роботи над вдосконаленням шаблону, написання статей, мануалів, створення тестових проектів у системах контрою версій
3.	Поглинання	Пропозиція купівлі розроблених технологій з метою її подальшого вдосконалення	Розвиток розроблених технологій у складі компанії-власника

6.4 Маркетингова концепція

Ступеневий аналіз конкуренції на ринку та аналіз конкуренції в галузі за М. Портером представлені в таблицях 10.8 та таблиці 10.9 відповідно. Конкурентний аналіз не спрямований на визначення можливостей, загроз і відшукування стратегічних невизначеностей, що можуть створюватися конкурентами, оскільки результати роботи знаходяться в відкритому доступі і питання конкуренції не постає.

Таблиця 6.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Олігополія	Відсутність конкурентів Велика ринкова сила Схожість використовуваних технологій	Інформування ринку щодо появи нового шаблону шляхом публікації статті
Внутрішньогалузева	Діяльність в одній галузі економіки Надання сервісів одного типу	Результати дослідження безкоштовні Збільшення кількості публікацій
Товарно-видова	Надання різних сервісів одного типу	Збільшення кількості публікацій
Цінова	Використання цін для покращення економічних умов збуту	Результати дослідження безкоштовні

Таблиця 6.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу		Висновки
Прямі конкуренти в галузі	Системи на основі статичних алгоритмів	Інтенсивна конкуренція, монополізація ринку збуту
Потенційні конкуренти	Розмір капіталовкладень,	Можливості входу на ринок забезпечить збільшення публікацій

6.5 Висновки до шостого розділу

Стартап проект підготовлений на базі даної магістерської роботи має всі шанси на високу комерціалізацію, за рахунок інноваційного підходу який наразі популярний в напрямках геймінгу, робототехніки та безпілотних автомобілів. Потенційний продукт має яскраво виражені сильні сторони такі, як наприклад, висока точність розпізнавання та швидкодія. Ринкова ситуація також виглядає досить сприятливою, за останні роки компанії вивели на ринок нові продукти та створили компанії єдинорогів. Тим не менше, на глобальному ринку мають місце кілька компаній, чия діяльність частково або суттєво схожа з функціоналом даного проекту, однак даний проект має свої переваги у порівнянні з конкурентами, а саме: точність розпізнавання та швидкодія. Що стосується маркетингової концепції та просування продукту, то тут варто виділити той факт, що даний продукт орієнтується на відносно вузький прошарок суспільства, а саме на представників середнього та великого бізнесу, а також державних органів.

ВИСНОВКИ

Глибинне навчання з підкріпленням наразі одне з найперспективніших напрямків штучного інтелекту, адже саме воно комбінує в собі останні досліджень в сфері нейронних мереж та реалізацією в динамічному середовищі, тобто в умовах найближчих до реальності.

В роботі було розглянута актуальна тема: застосування методів інтелектуального аналізу даних для класифікації зображення та навчання Q функції в методах навчання з підкріпленням на прикладі гри Minecraft.

Поставлені задачі:

1. Провести дослідження існуючих підходів класифікації зображення.
2. Дослідити та вивчити основні принципи роботи методики машинного навчання Q-Learning.
3. Підібрати навчальну та тестову вибірку на основі знімків з гри Minecraft.
4. Спроектувати та побудувати програмний комплекс, що реалізував автоматичну класифікацію предметів.
5. Розробити тестове середовище для оцінки політики агента на основі вихідних даних після класифікації зображення.

Під час виконання поставлених задач проведено:

- аналіз знімків з гри Minecraft на предмет придатності для розпізнання об’єктів;
- дослідження сучасних підходів у класифікації зображень а також в поєднанні з навчанням з підкріпленням;
- підбір раціональної вибірки та обробка набору даних для випробувань моделей;

- побудова моделей класифікації на основі методів опорних векторів, наївного байєсівського класифікатора та дерев рішень;
- проектування та реалізація архітектури згорткової нейронної мережі для класифікації зображень;
- аналіз кожної з моделей по ключовим метрикам;
- розробка програмного комплексу що реалізує класифікацію зображень для оцінки політики агента в глибокому навчанні з підкріпленням на прикладі гри Minecraft;
- дослідження найкращої моделі на предмет стартап- проекту;

В результаті було отримано модель нейронної мережі котра здатна розпізнавати десять класів з оцінкою повноти 0.87 та F-мірою 0.83, та передавати результати агенту для оцінки наступного кроку для максимізації виграшу.

В майбутньому рекомендується продовжити дослідження в даному напрямі:

- удосконалення моделі класифікації та архітектури нейронної мережі, випробувати ансамбль підходів;
- збільшити кількість класів для класифікації;
- використати моделі та підходи для схожих задач в динамічному середовищі, наприклад в сфері робототехніки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Gemp I., Theocharous G., Ghavamzadeh M. Association for the Advancement of Artificial Intelligence. People.cs. 2017. URL: https://people.cs.umass.edu/~imgemp/pubs/iaai_2017.pdf
2. Sequeira P. Aglomera.NET. *GitHub*. URL: https://github.com/pedrodbs/Aglomera
3. Camargos R. C. Finding groups in data with C# – Agglomerative Clustering. *Code Project*. 2016. URL: https://www.codeproject.com/Articles/1120804/Finding-groups-in-data-with-Csharp-Agglomerative-C
4. Doxakis P. HdbscanSharp. *GitHub*. URL: https://github.com/doxakis/HdbscanSharp
5. Eldridge J., Belkin M., Wang Y. Beyound Hartigan Consistency: Merge Distortion Metric for Hierarchical Clustering. *arXiv*. 2015. URL: https://arxiv.org/pdf/1506.06422v2.pdf
6. How HDBSCAN Works. *Jupiter nbviewer*. URL: https://nbviewer.jupyter.org/github/scikit-learn-contrib/hdbscan/blob/master/notebooks/How%20HDBSCAN%20Works.ipynb
7. Rodriguez A., Laio A. Clustering by fast search and find of density peaks. *Science*. 2014. Vol. 344, Issue 6191. Pp. 1492-1496. URL: https://science.sciencemag.org/content/sci/344/6191/1492.full.pdf
8. The Wirecutter. What is Alexa? What is the Amazon Echo, and should you get one? URL: https://thewirecutter.com/reviews/what-is-alexa-what-is-the-amazon-echo- and-should-you-get-one/
9. Bresnick J. How to Choose the Right Healthcare Big Data Analytics Tools . URL: https://healthitanalytics.com/features/how-to-choose-the-right-healthcare- big-data-analytics-tools

10. Bresnick J. Imaging Analytics Get Big Data Boost from New Partnerships. URL: https://healthitanalytics.com/news/imaging-analytics-get-big-data-boost-from-new-partnerships
11. Bresnick J. IBM Watson Expands Role in Imaging Analytics, Health. URL: https://healthitanalytics.com/news/ibm-watson- analytics-population-health
12. Bresnick J. Machine Learning, NLP Help with Physician Skill Benchmarking. URL:https://healthitanalytics.com/news/machine-learning-nlp-help-with-physician-skill-benchmarking
13. Bresnick J. UCSF to Develop Machine Learning for CDS, Imaging. URL: https://healthitanalytics.com/news/ucsf-to-develop-machine-learning-for- cds-imaging-analytics
14. Rajpurkar P., Hannun A., Haghpanahi M., Bourn C., NG A.Y. Stanford. URL: https://arxiv.org/pdf/1707.01836.pdf
15. Predicting Response to Depression Treatment (PReDicT) project. URL: http://p1vital.com/ehealth/?p=376
16. Payan A. Predicting Alzheimer's disease: a neuroimaging study with 3D convolutional neural networks. URL: https://arxiv.org/abs/1502.02506
17. Mukherjee S. A.I. VERSUS M.D. What happens when diagnosis is automated?. URL: https://www.newyorker.com/magazine/2017/04/03/ai-versus-md
18. Арустамов А. Статья: Предобработка и очистка данных перед загрузкой в хранилище. URL: http://sysdba.org.ua/proektirovanie-bd/etl/predobrabotka-i- ochistka-dannyih-pered-zagruzkoy-v-hranilische.html
19. Фоурино Р. Электронное качество данных: скрытая перспектива очистки данных. URL: http://www.iso.ru/print/rus/document5820.phtml
20. Воронцов К.В. Машинное обучение: курс лекций. URL: http://www.machinelearning.ru/wiki/index.php?
21. Воронцов К.В. Лекции по логическим алгоритмам классификации. URL: http://www.ccas.ru/voron/download/LogicAlgs.pdf

22. Курс "Машинное обучение" на ФКН ВШЭ. URL: https://github.com/esokolov/ml-course-hse
23. Rodrıguez J.J. Rotation Forest: A New Classifier Ensemble Method. URL: https://ieeexplore.ieee.org/abstract/document/1677518/
24. Дьяконов А. Введение в анализ данных и машинное обучение. URL: https://alexanderdyakonov.files.wordpress.com/2017/06/book_boosting_pdf.pdf
25. Chen T. XGBoost: A Scalable Tree Boosting System/ Chen T., Guestrin C. . URL: https://arxiv.org/abs/1603.02754
26. Tumer K.A Error Correlation and Error Reduction in Ensemble Classiers. URL: https://www.tandfonline.com/doi/abs/10.1080/095400996116839
27. Hand D.J. Measuring classifier performance: a coherent alternative to the area under the ROC curve. URL: https://link.springer.com/content/pdf/10.1007%2Fs10994-009-5119-5.pdf
28. Hern'andez-Orallo J. Brier Curves: A New Cost-Based Visualisation of Classifier Performance. URL: https://pdfs.semanticscholar.org/2bc1/743405cf276125e798fbd96290b518c51d56.pdf
29. Sokolova M. Beyond Accuracy, F-score and ROC: a Family of Discriminant Measures for Performance Evaluation. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.549.2795&rep=rep1&type=pdf
30. Youden, W. Index for rating diagnostic tests. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/10970142%281950%293%3A1%3C32%3A%3AAID- CNCR2820030106%3E3.0.CO%3B2-3

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

```
from commands.BaseCommand import BaseCommand
import xgboost as xgb
from sklearn.model_selection import train_test_split
from config import env
from library import helpers
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
import time
import joblib
import json
import os
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import pandas as pd


class DayPredictModelBinary(BaseCommand):
    """
    Task model

    click_predict:predict_day_model_binary
    """

    path = None
    coeff_unbalanced = None
    day = None

    def handle(self):
        for day in range(1, 8):
            self.day = day
            print(self.day)
            self.path = helpers.init_dir(env.APP_DIR + "cache/clickPredict/")
            dataset = helpers.read_csv_part(self.path + 'Kismia_day{0}.csv'.format(self.day), 500)
            dataset = helpers.delete_column_list(dataset, (0,0))
            data, label, names = self.prepare_dataset(dataset)
            # model, dtrain, dtest, y_train, y_test = self.model(data, label, names)
            # predict = self.model_predict(model, dtest)
            # self.predict_info(model, predict, dtest, y_test)
            # self.safe_model(model, y_train)
            print('done')

    def prepare_dataset(self, dataset):

        label = []
        data = []
```

```
        names = dataset[0]
        dataset = dataset[1:]

        names.pop(84)
        positive = 0

        for row in dataset:
            data_row = []
            for k, v in enumerate(row):
                if k == 84:
                    if np.int(float(v)) > 0:
                        v = 1
                        positive += 1
                    label.append(np.int(float(v)))
                else:
                    data_row.append(np.int(float(v)))
            data.append(data_row)
        negative = len(dataset)-positive
        self.coeff_unbalanced = negative/positive
        print('positive is {0} and neg is {1}, coef {2}'.format(positive, negative,
int(self.coeff_unbalanced)))

        return data, label, names

    def model(self, data, label, names):

        X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.33,
random_state=42)
        feature_name = names

        dtrain = xgb.DMatrix(X_train, y_train, feature_names=feature_name, missing=999)
        dtest = xgb.DMatrix(X_test, y_test, feature_names=feature_name, missing=999)


        print("Train dataset contains {0} rows and {1} columns".format(dtrain.num_row(),
dtrain.num_col()))
        print("Test dataset contains {0} rows and {1} columns".format(dtest.num_row(),
dtest.num_col()))

        params = {
            'objective': 'binary:logistic',
            'max_depth': 7,
            'silent': 1,
            'eta': 0.1,
            'scale_pos_weight': int(self.coeff_unbalanced)

        }

        num_rounds = 130
        evallist = [(dtest, 'eval'), (dtrain, 'train')]

        model = xgb.train(params, dtrain, num_rounds, evals=evallist)
```

```
        return model, dtrain, dtest, y_train, y_test

    def model_predict(self, model, dtest):

        predict = model.predict(dtest, ntree_limit=model.best_ntree_limit)
        # to csv file for analysis
        # prediction = pd.DataFrame(predict, columns=['predictions']).to_csv(self.path +
"prediction.csv")

        return predict

    def predict_info(self, model, predict, dtest, y_test):

        predicted_labels = predict > 0.5

        print('Accuracy: {0:.2f}'.format(accuracy_score(dtest.get_label(), predicted_labels)))
        print('Precision: {0:.2f}'.format(precision_score(dtest.get_label(), predicted_labels)))
        print('Recall: {0:.2f}'.format(recall_score(dtest.get_label(), predicted_labels)))
        print('F1: {0:.2f}'.format(f1_score(dtest.get_label(), predicted_labels)))

        importances = model.get_fscore()
        print(importances)

        xgb.plot_importance(model)
        plt.savefig(self.path + 'taskBinary')
        print('saved')

    def safe_model(self, model, y_train):

        time_start = time.time()

        # Making list from ytrain for hashing
        train_list = []
        for row in y_train:
            train_list.append(row)
        train_list.sort()
        train_hash = helpers.hash_crc32(json.dumps(train_list))
        print('model hash: ' + train_hash)

        # Building unique cache file name
        # cache_file = self.path + 'model/' + 'Kismia_Day1' + '.gz'
        cache_model = self.path + 'model/' + 'Kismia_Day{0}'.format(self.day) + '.model'

        # Check if already cached
        # if os.path.exists(cache_file):
        #     return joblib.load(cache_file)

        # Caching
        # joblib.dump(model, cache_file, 9)
        model.save_model(cache_model)
        print('model in %s' % (time.time() - time_start))
```

```
class ClassifModel(BaseCommand):
    """
    Classification Model

    classif:build_model
    """

    def handle(self):
        articles =
helpers.read_csv('/home/ubuntu/www/python/data/classifText/rj_classif_data_lable.csv')
        data, label, article_id = self.prepare_dataset(articles)
        data = self.text_cleaning(data)
        self.buil_model(data, label)

    def text_cleaning(self, articles):
        data = []
        for article in articles:
            article = prepare.clear_list(article)
            article = prepare.stopwords_list(article)
            article = prepare.stem_list(article)
            data.append(''.join(article) )
        # joblib.dump(articles, '/home/ubuntu/www/python/data/clusterText/rj_classif.gz')
        return data

    def prepare_dataset(self, dataset):

        article_id = []
        label = []
        data = []
        for row in dataset:
            data_row = []
            for k, v in enumerate(row):
                if k == 0:
                    article_id.append(int(v))
                if k == 1:
                    data_row.append(v)
                if k == 2:
                    label.append(int(v))
            data.append(data_row)

        return data, label, article_id

    def buil_model(self, data, label):
        X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.33,
random_state=42)

        model = models.build_classification(X_train, y_train)
        predicted = model.predict(X_test)
        accuracy = accuracy_score(predicted, y_test)
```

```
        print(accuracy)
        print(predicted)

def gender(gender_id):
    if gender_id == 'm':
        return 1
    elif gender_id == 'f':
        return 2
    return 3


def feature_list(user_list: dict, features):
    users_features = []
    feature_name = []

    for feature in features:
        if feature not in user_list:
            continue
        f = user_list[feature]
        if f == None:
            f = 999
        users_features.append(int(f))
        feature_name.append(feature)

    if len(features) != len(feature_name):
        print('not full feature list')

    return users_features, feature_name


def feature_dictionary(user_data: dict, features):
    users_features = {}

    for feature in features:
        if feature not in user_data.keys():
            continue

        f = user_data[feature]
        if f is None:
            f = 999
        users_features[feature] = f

    return users_features


def ageGroup(age):
    if age is None:
        return 999
    if age < 30:
        return 1
    if age >= 30 and age < 40:
        return 2
```

```
    if age >= 40 and age < 50:
        return 3
    if age >= 50 and age < 70:
        return 4
    if age >= 70 and age < 80:
        return 5
    else:
        return 6


def date_part_datetime(date_datetime):
    if date_datetime is None:
        return 999
    return int(date_datetime.isoweekday())


def date_part_timestamp(date_timestamp):
    if date_timestamp is None:
        return 999
    return int(datetime.datetime.fromtimestamp(int(date_timestamp)).isoweekday())


def date_diff_now(date):
    if date is None:
        return 999
    return (datetime.datetime.now() - date).days


def date_diff_now_timestamp(lr):
    if lr is None:
        return 999
    else:
        return (datetime.datetime.now() - datetime.datetime.fromtimestamp(lr)).days


class Graylog():
    """
    :type __graylogHost__: str
    :type __graylogPort__: int
    :type __graylogCodebase__: str
    """

    __graylogHost__ = None
    __graylogPort__ = None
    __graylogCodebase__ = None

    def __init__(self, graylogHost, graylogPort, graylogCodebase):
        self.__graylogHost__ = graylogHost
        self.__graylogPort__ = graylogPort
        self.__graylogCodebase__ = graylogCodebase

    def connect(self):
```

```
        logger = logging.getLogger(self.__graylogCodebase__)
        logger.setLevel(logging.DEBUG)

        handler = graypy.GELFUDPHandler(self.__graylogHost__, self.__graylogPort__)
        logger.addHandler(handler)
        return logger

def read_csv(file):
    with open(file, 'r', encoding='utf-8') as f:
        reader = csv.reader(f)
        return list(reader)


def read_csv_part(file, cut=0):
    with open(file, 'r', encoding='utf-8') as f:
        line = 0
        data = []
        while line < cut:
            line += 1
            data.append(f.readline())
        reader = csv.reader(data)
        return list(reader)


def read_csv_column(file, column_id):
    result = []
    data = read_csv(file)
    for row in data:
        if len(row) - 1 >= column_id:
            result.append(row[column_id])
    return result


def read_file(file, limit=0):
    if not os.path.exists(file):
        return False
    with open(file, 'r', encoding='utf-8') as myfile:
        if limit:
            result = []
            lineNum = 0
            for line in myfile:
                lineNum += 1
                result.append(line)
                if lineNum >= limit:
                    break
            return result
        else:
            return myfile.read()


def hash_crc32(string):
    return "%x" % zlib.crc32(string.encode('utf-8'))
```

```
def write_all_file(file, text):
    text = str(text)
    with open(file, 'w', encoding='utf-8') as the_file:
        return the_file.write(text)


def get_html_by_link(link: str) -> str:
    ctx = ssl.create_default_context()
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE
    try:
        fp = urllib.request.urlopen(link, context=ctx)
        mybytes = fp.read()
        mystr = mybytes.decode("utf8")
        fp.close()
    except HTTPError:
        return ''
    except URLError:
        return ''
    except ValueError:
        return ''
    else:
        return mystr


def delete_column_list(list, columns: tuple):
    for row in list:
        for index in columns:
            row.pop(index)
    return list


def init_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)
    return path


def prepareString(value):
    if value is None:
        value = ''
    return str(value)


def find_in_lines(phrase, listOfLines):
    for row in listOfLines:
        if row.find(phrase) != -1:
            return True
    return False
```